Projecting Assumptions: The Duality Between Sparse Autoencoders and Concept Geometry

Sai Sumedh R. Hindupur [*1] Ekdeep Singh Lubana [*2] Thomas Fel [*3] Demba Ba [1 3]

Abstract

Sparse Autoencoders (SAEs) are widely used to interpret neural networks by identifying meaningful concepts from their representations. We show that each SAE imposes structural assumptions about how concepts are encoded in model representations, which in turn shapes what it can and cannot detect. We train SAEs on synthetic data with specific structure to show that SAEs fail to recover concepts when their assumptions are ignored, and we design a new SAE—called SpaDE—that enables the discovery of previously hidden concepts (those with heterogenous intrinsic dimensionality and nonlinear separation boundaries) and reinforces our theoretical insights.

1. Introduction

Interpretability has become an important research agenda for assuring, debugging, and controlling neural networks (Anwar et al., 2024; Bengio et al., 2025; Lehalleur et al., 2025; Rudin et al., 2022; Adebayo et al., 2020). To this end, sparse dictionary learning methods (Serre, 2006; Faruqui et al., 2015; Subramanian et al., 2018; Arora et al., 2018; Olshausen & Field, 1996a), especially Sparse Autoencoders (SAEs), have seen a resurgence in literature, since they offer an unsupervised pipeline for simultaneously enumerating all concepts a model may rely on for making its predictions (Cunningham et al., 2023; Bricken et al., 2023; Gao et al., 2024; Rajamanoharan et al., 2024b; Fel et al., 2025; Bussmann et al., 2024; Fel et al., 2023; Colin et al., 2024). Specifically, an SAE decomposes representations into an overcomplete set of latents that (ideally) correspond to abstract, data-centric concepts which, upon aggregation, explain away the model representations (Kim et al., 2018; Fel, 2025). In other words, an SAE is expected to result in *monosemantic* latents which are more interpretable than the neurons of the original model (Elhage et al., 2022).

In this work, we show that different SAEs have different abilities to extract concepts, beyond just in their fidelity/sparsity. We show that any SAE is implicitly biased towards identifying concepts as monosemantic that are organized in a specific manner (Fig. 1) using SAE latent receptive fields (see Sec. 2), and highlight the assumptions for popular SAEs. We evaluate SAEs on concepts which violate their assumptions through experiments on controlled synthetic setups, demonstrating that SAEs failing to account for these properties systematically miss the corresponding concepts. We introduce SpaDE (Sparsemax Distance Encoder), a novel SAE that explicitly incorporates data properties other SAEs cannot capture into its encoder. As we show, SpaDE successfully identifies concepts that other SAEs fail to detect, reinforcing the need for data-aware choices in interpretability.

2. Preliminaries

Notation. We denote vectors as lowercase bold (e.g., $\boldsymbol{x}$) and matrices as uppercase bold (e.g., $\boldsymbol{X}$). $[n]$ denotes $\{1, \ldots, n\}$ and $\mathcal{B} = \{\boldsymbol{x} \mid \|\boldsymbol{x}\|_2 \leq 1\}$ the unit ℓ_2-ball in $\mathbb{R}^d$. We assume access to a dataset of k samples, $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, where $\boldsymbol{x} \in \mathbb{R}^d$. For any matrix $\boldsymbol{X}$ or vector $\boldsymbol{x}$, we use $\boldsymbol{X} \geq \boldsymbol{0}$ (resp. $\boldsymbol{x} \geq \boldsymbol{0}$) to indicate element-wise non-negativity.

Sparse Coding. Also known as Sparse Dictionary Learning (Olshausen & Field, 1996a; 1997), sparse coding assumes a generative model of data as a sparse combination of latents. Specifically, sparse coding involves solving the following optimization problem: $\arg\min_{\boldsymbol{z} \geq \boldsymbol{0}, \boldsymbol{D} \in \mathcal{B}} \quad \sum_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \lambda \mathcal{R}(\boldsymbol{z})$, where $\boldsymbol{z} \in \mathbb{R}^s$ is a sparse latent code, $\boldsymbol{D} \in \mathbb{R}^{d \times s}$ are

[*] Equal contribution [1] School of Engineering and Applied Science, Harvard University [2] CBS-NTT Program in Physics of Intelligence, Harvard University [3] Kempner Institute, Harvard University. Correspondence to: Sai Sumedh R. Hindupur <shindupurravindra@g.harvard.edu>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

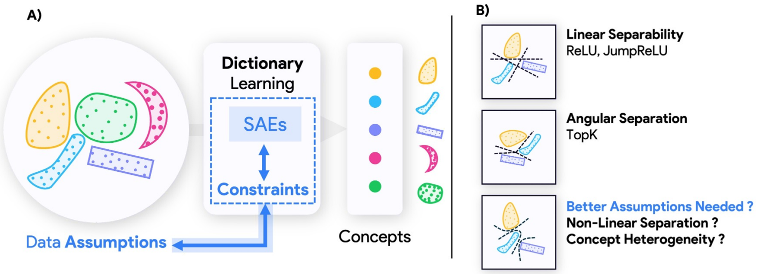

Figure 1: **The Duality Between SAEs Architectures and Their Implicit Data Assumptions. A)** SAEs do not passively extract concepts—they impose constraints that shape what can be detected. **B)** Different SAEs rely on different assumptions: some expect features to be linearly separable (ReLU, JumpReLU) or separable by angle while having uniform intrinsic dimensionality (TopK).

the dictionary atoms, and $\mathcal{R}(\boldsymbol{z})$ is a sparsity-promoting regularizer, typically $\|\boldsymbol{z}\|_1$. Note that the optimization is performed over *both* the sparse code $\boldsymbol{z}$ (with $\boldsymbol{z} \geq \mathbf{0}$) and the dictionary $\boldsymbol{D}$. Further details are included in Appendix A.

Sparse Autoencoders. SAEs (Ng et al., 2011) approximate sparse dictionary learning by using a single hidden layer to compute the sparse code from data ($x \in \mathbb{R}^d$): (i) $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}_e)$, and (ii) $\hat{\boldsymbol{x}} = \boldsymbol{D}\boldsymbol{z} + \boldsymbol{b}_d$, where $\boldsymbol{W}, \boldsymbol{D} \in \mathbb{R}^{d \times s}$ and $\boldsymbol{g} : \mathbb{R}^s \to \mathbb{R}^s$ is the encoder non-linearity. Here, sparsity is enforced on the SAE latent code $\boldsymbol{z}$. SAEs are trained on the sparse dictionary learning loss, with the sparsity-promoting regularizer $\mathcal{R}$. Different SAEs typically differ in the choice of encoder nonlinearity $\boldsymbol{g}$ and the regularizer $\mathcal{R}$.
We use the concept of receptive fields from neuroscience to highlight monosemanticity properties of SAEs.

Definition 2.1 (Receptive Field)**.** The receptive field of a neuron k, which computes a function $f^{(k)} : \mathbb{R}^d \to \mathbb{R}$, is defined as $\mathcal{F}_k = \{\boldsymbol{x} \in \mathbb{R}^d \mid f^{(k)}(\boldsymbol{x}) > 0\}$. Intuitively, $\mathcal{F}_k$ represents the region of input space where neuron k is active.

3. Implicit SAE Assumptions and Data Properties

In this section, we explicitly state the data assumptions made by ReLU SAE (Cunningham et al., 2023; Bricken et al., 2023), TopK SAE (Gao et al., 2024; Makhzani & Frey, 2013) and JumpReLU SAE (Rajamanoharan et al., 2024b).

Theorem 3.1 (Implicit Assumptions; Informal)**.** *An SAE makes implicit assumptions about the structure of concepts in data, reflecting it in the receptive fields of its encoder. These assumptions are explicitly stated in Tab. 1 for ReLU, JumpReLU and TopK SAEs (derived in App. E.2).*

The optimality of the above assumptions depends on the "true structure" of concepts in model representations. While concept structure in not known in its entirety, we highlight two properties of how (certain) concepts are organized in a model based on recent interpretability literature.

1. **Nonlinear separability of concepts.** Concepts are not separable by linear decision boundaries. Evidence towards such concepts include features with dependence on magnitude, such as onion features (Csordás et al., 2024). Even "linear features" (Arora et al., 2018; Park et al., 2023)), having different magnitudes may fail to be linearly separable (Fig. 2).
2. **Heterogeneity of concepts.** Different concepts belong to subspaces with different dimensions. Evidence for this property includes unidimensional features representable as concept activation vectors (Kim

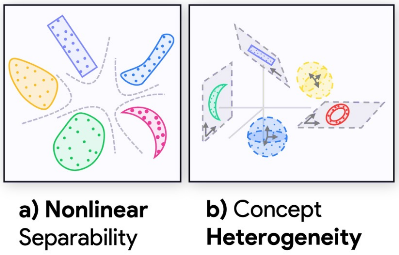

Figure 2: **Illustration of Two Reasonable Data Assumptions. A)** Concepts may not be separable using hyperplanes. **B)** Some concepts are inherently low-dimensional, while others span higher-dimensional spaces.

Table 1: **Implicit Assumptions of SAEs.** The receptive fields of SAEs implicitly assume concepts are organized with a specific structure in the data, i.e., in model representations.

Model	Receptive Field	Data Assumption
ReLU	half-spaces	Linear separability of concepts
JumpReLU	half-spaces	Linear separability of concepts
TopK	union of hyperpyramids	Angular separability of concepts; same dimensionality per concept

et al., 2018), e.g., truth (Bürger et al., 2025), multidimensional features such as days of the week in a 2-D subspace (Engels et al., 2024a), and higher dimensional safety-relevant features (Pan et al., 2025).

We characterize the compatibility of different SAEs' implicit assumptions and these concept properties in Tab. 2. To enable evaluation of our claims, we next design an SAE that accommodates the two properties above into its architecture, presented in the following subsection.

Table 2: **Compatibility of SAEs** with nonlinear separability and heterogeneity.

Model	Nonlinear Sep.	Heterogeneity
ReLU	✗	✓
JumpReLU	✗	✓
TopK	✓	✗

3.1. SpaDE: Designing a Geometry-driven SAE

We now use the data properties studied above—nonlinear separability and concept heterogeneity—and through the duality, construct one set of sufficient conditions on the SAE to capture both properties, resulting in a novel SAE called SpaDE (Sparsemax Distance Encoder). While details are provided in App. E.4, we note SpaDE (**Spa**rsemax **D**istance **E**ncoder) is a combination of the Sparsemax ((Martins & Astudillo, 2016)) function with euclidean distances:

$$\begin{gathered} \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \text{Sparsemax}(-\lambda d(\boldsymbol{x}, \boldsymbol{W})), \\ \text{where } d(\boldsymbol{x}, \boldsymbol{W})_i = \|\boldsymbol{x} - \boldsymbol{W}_i\|_2^2, \end{gathered} \tag{1}$$

$$\text{Sparsemax}(\boldsymbol{v}) = \underset{\boldsymbol{\pi} \in \Delta^s}{\arg\min} \|\boldsymbol{\pi} - \boldsymbol{v}\|_2^2.$$

In the above, λ is a scaling parameter (akin to inverse temperature), while $\boldsymbol{W}_i$ is the i^{th} column of the encoder matrix $\boldsymbol{W}$ which behaves as a *prototype* (or landmark) in input space since we compute euclidean distance from input $\boldsymbol{x}$ to $\boldsymbol{W}_i$. The regularizer for SpaDE is a distance-weighted ℓ_1 regularizer $\mathcal{R}(\boldsymbol{z}) = \sum_i z_i \|\boldsymbol{x} - \boldsymbol{W}_i\|_2^2$ (KDS, (Tasissa et al., 2023))[1]. App. E.4 and E.2.3 describe the receptive fields of SpaDE in further detail and show how it captures nonlinear separability and concept heterogeneity.

4. Results: Empirical Validation of SAE behavior

We perform experiments by training ReLU, JumpReLU, TopK and SpaDE SAEs on synthetic Gaussian clusters. Further analysis is in App. F. We include experiments on more naturalistic data (formal language models, vision models) in App. F.

4.1. Separability Experiment

Dataset and Experiment: We construct a 2-dimensional dataset with Gaussian clusters (abstraction of concepts) of different magnitudes in order to demonstrate nonlinear separability of concepts in a simple setting which facilitates visualization. The concepts with smaller norm are not linearly separable, while those with larger norm are linearly separable. Following our arguments about implicit assumptions in SAEs, we hypothesize that ReLU and JumpReLU SAEs will be unable to capture the nonlinearly separable concepts with monosemantic latents (measured using F1 scores; see Eq. 6).

Observations: Fig. 3 shows how different SAEs fare on this experiment. In the receptive fields of Row (b), ReLU and JumpReLU show monosemantic latents for the separable concept (orange), but latents activating for multiple concepts for the nonlinearly separable case (purple). This is quantified with F1 scores in Row (a), and further through within- and cross-concept co-occurrence of SAE latents (Row (c)). SpaDE shows flexible receptive fields, top F1 scores and no cross-concept correlations.

4.2. Heterogeneity Experiment

Dataset and Experiment: We generate Gaussian clusters (again an abstraction for concepts) in a 128-dimensional space. The five concepts are heterogeneous—they belong to subspaces with different intrinsic dimensions (6, 14, 30, 62, 126), but are designed to have isotropic structure within each cluster, and similar total variances across clusters. We trained ReLU,

[1] This regularizer encourages dictionary atoms to "stick" to the data, addressing the recently raised concern (Fel et al., 2025; Paulo & Belrose, 2025) that directions learned by SAEs may be out-of-distribution (OOD), contributing to their instability.

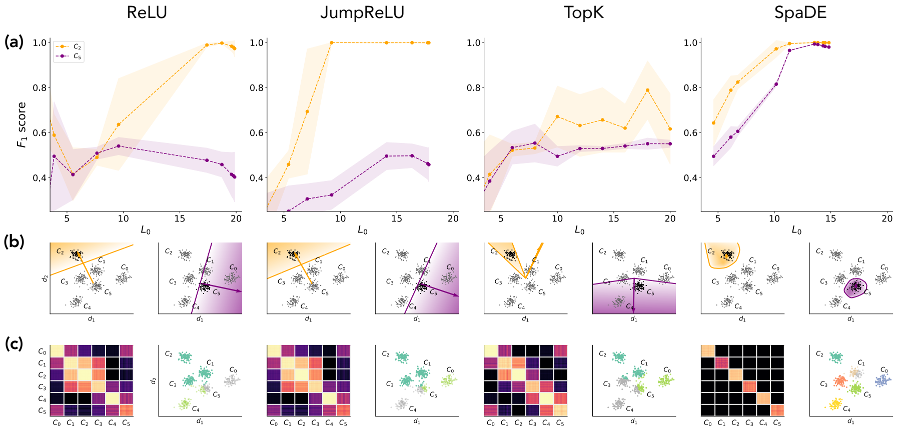

Figure 3: **Effect of Nonlinear Separability on SAEs**. Each column represents a different SAE. **a)** F_1 scores of the top 5 most monosemantic latents (highest F1 scores), where shaded region is ± 1SD, of each SAE on two concepts—orange (linearly separable) and purple (non-linearly separable). **b)** Receptive fields of the most monosemantic latent for each SAE. Intensity of color indicates strength of SAE latent activation. **(c)** Matrix of pairwise cosine similarities between sparse codes of different datapoints, and data clusters obtained through spectral clustering on this matrix.

JumpReLU, TopK SAEs and SpaDE on this data with varying sparsity levels. We hypothesize that TopK will not be able to adapt its representations to the intrinsic dimension of each cluster.

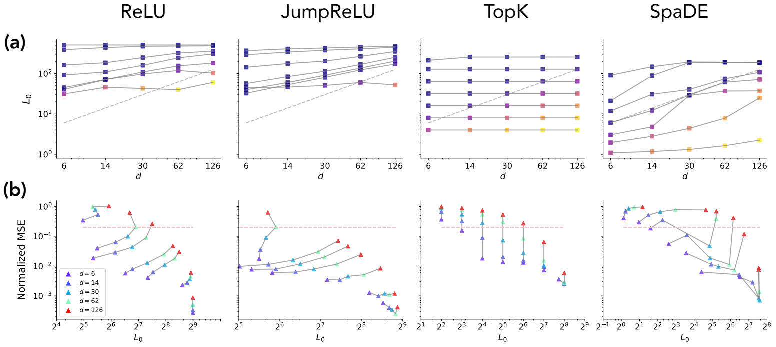

Figure 4: **Effect of Concept Heterogeneity on SAEs**. **a)** Per-concept sparsity as a function of intrinsic dimension. Colors indicate per-concept MSE—higher errors (red/yellow) show when an SAE fails to capture a concept effectively. Each solid line indicates one model with a specific choice of hyperparameters. **b)** Normalized MSE vs. per-concept sparsity. TopK SAE only achieves good reconstruction (below the dashed 20% error threshold) only when K exceeds intrinsic dimensionality.

Observations: Fig. 4 shows the results of all SAEs on this experiment. In Row (a), TopK shows the same level of sparsity per concept for all concepts, along with worse reconstruction error for higher dimensional concepts. For TopK, normalized MSE (Row (b)) goes below 20% (i.e., explains 80% of the variance) for each concept only when k exceeds the dimension of that concept. In contrast, other SAEs—ReLU, JumpReLU and SpaDE—show adaptive sparsity to different extents, and stay below the 20% threshold for nearly all concepts across hyperparameters.

5. Discussion and Limitations

Our observations about the limitations of ReLU, JumpReLU and TopK SAEs highlight that the failure modes of different SAEs stem from a mismatch between their inductive biases and the true structure of the data. This suggests the interpretability community may need to prioritize a deeper understanding of latent space geometry, and translate novel insights into SAE design, leading to models with more faithful and structured representations of concepts.

Limitations: We present SpaDE as a concrete example of incorporating reasonable data properties (nonlinear separability and concept heterogeneity) into SAE design. Data properties beyond those considered here may be crucial for improved SAE performance. SpaDE implicitly assumes concepts are separated by Euclidean distance, which may still result in latent co-occurrence if concepts do not satisfy this assumption. We have focused our attention on mutually exclusive concepts in this work, where the presence of one concept implies the absence of others.

Acknowledgments

ESL thanks Hidenori Tanaka and the Physics of Intelligence group at CBS-NTT program Harvard for useful conversations and access to compute resources. SSRH thanks the CRISP group at Harvard SEAS for insightful conversations, and the Kempner Institute at Harvard University for access to compute resources.

References

Adams, E., Bai, L., Lee, M., Yu, Y., and AlQuraishi, M. From mechanistic interpretability to mechanistic biology: Training, evaluating, and interpreting sparse autoencoders on protein language models. *bioRxiv*, pp. 2025–02, 2025.

Adebayo, J., Muelly, M., Liccardi, I., and Kim, B. Debugging tests for model explanations. *arXiv preprint arXiv:2011.05429*, 2020.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*, 2023.

An, B., Wang, S., Zhao, Z., Qin, F., Yan, R., and Chen, X. Interpretable neural network via algorithm unrolling for mechanical fault diagnosis. *IEEE Transactions on Instrumentation and Measurement*, 71:1–11, 2022.

Andrej Karpathy. nanoGPT, 2023. https://github.com/karpathy/nanoGPT.

Anwar, U., Saparov, A., Rando, J., Paleka, D., Turpin, M., Hase, P., Lubana, E. S., Jenner, E., Casper, S., Sourbut, O., et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*, 2024.

Arora, S., Li, Y., Liang, Y., Ma, T., and Risteski, A. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

Ayonrinde, K., Pearce, M. T., and Sharkey, L. Interpretability as compression: Reconsidering sae explanations of neural activations with mdl-saes. *arXiv preprint arXiv:2410.11179*, 2024.

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Bengio, Y., Mindermann, S., Privitera, D., Besiroglu, T., Bommasani, R., Casper, S., Choi, Y., Fox, P., Garfinkel, B., Goldfarb, D., et al. International ai safety report. *arXiv preprint arXiv:2501.17805*, 2025.

Bhalla, U., Srinivas, S., Ghandeharioun, A., and Lakkaraju, H. Towards unifying interpretability and control: Evaluation via intervention. *arXiv preprint arXiv:2411.04430*, 2024.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Bürger, L., Hamprecht, F. A., and Nadler, B. Truth is universal: Robust detection of lies in llms. *Advances in Neural Information Processing Systems*, 37:138393–138431, 2025.

Bussmann, B., Leask, P., and Nanda, N. Batchtopk sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024.

Chen, S., Eldar, Y. C., and Zhao, L. Graph unrolling networks: Interpretable neural networks for graph signal denoising. *IEEE Transactions on Signal Processing*, 69:3699–3713, 2021.

Colin, J., Goetschalckx, L., Fel, T., Boutin, V., Gopal, J., Serre, T., and Oliver, N. Local vs distributed representations: What is the right basis for interpretability? *arXiv preprint arXiv:2411.03993*, 2024.

Csordás, R., Potts, C., Manning, C. D., and Geiger, A. Recurrent neural networks learn to store and generate sequences using non-linear representations, 2024. URL https://arxiv.org/abs/2408.10920.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Daubechies, I., Defrise, M., and De Mol, C. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Dunefsky, J., Chlenski, P., and Nanda, N. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2025.

Durmus, E., Tamkin, A., Clark, J., Wei, J., Marcus, J., Batson, J., Handa, K., Lovitt, L., Tong, M., McCain, M., et al. Evaluating feature steering: A case study in mitigating social biases, 2024. *URL https://anthropic. com/research/evaluating-feature-steering*, 2024.

Elad, M., Milanfar, P., and Rubinstein, R. Analysis versus synthesis in signal priors. *Inverse problems*, 23(3):947, 2007.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. Toy models of superposition. *Transformer Circuits Thread*, 2022.

Engels, J., Michaud, E. J., Liao, I., Gurnee, W., and Tegmark, M. Not all language model features are linear, 2024a. URL https://arxiv.org/abs/2405.14860.

Engels, J., Riggs, L., and Tegmark, M. Decomposing the dark matter of sparse autoencoders. *arXiv preprint arXiv:2410.14670*, 2024b.

Faruqui, M., Tsvetkov, Y., Yogatama, D., Dyer, C., and Smith, N. Sparse overcomplete word vector representations. *arXiv preprint arXiv:1506.02004*, 2015.

Fel, T. Sparks of explainability: Recent advancements in explaining large vision models. *arXiv preprint arXiv:2502.01048*, 2025.

Fel, T., Picard, A., Bethune, L., Boissin, T., Vigouroux, D., Colin, J., Cadène, R., and Serre, T. Craft: Concept recursive activation factorization for explainability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2711–2721, 2023.

Fel, T., Lubana, E. S., Prince, J. S., Kowal, M., Boutin, V., Papadimitriou, I., Wang, B., Wattenberg, M., Ba, D., and Konkle, T. Archetypal sae: Adaptive and stable dictionary learning for concept extraction in large vision models, 2025. URL https://arxiv.org/abs/2502.12892.

Gao, L., la Tour, T. D., Tillman, H., Goh, G., Troll, R., Radford, A., Sutskever, I., Leike, J., and Wu, J. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Garcia, E. N. V. and Ansuini, A. Interpreting and steering protein language models through sparse autoencoders. *arXiv preprint arXiv:2502.09135*, 2025.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on machine learning*, pp. 399–406, 2010.

Gresele, L., Rubenstein, P. K., Mehrjou, A., Locatello, F., and Scholkopf, B. The incomplete rosetta stone problem: Identifiability results for multi-view nonlinear ica. *Uncertainty in Artificial Intelligence*, 2020.

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. 2017.

Jain, S., Kirk, R., Lubana, E. S., Dick, R. P., Tanaka, H., Grefenstette, E., Rocktäschel, T., and Krueger, D. S. Mechanistically analyzing the effects of fine-tuning on procedurally defined tasks. *arXiv preprint arXiv:2311.12786*, 2023.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pp. 2668–2677. PMLR, 2018.

Kissane, C., Krzyzanowski, R., Nanda, N., and Conmy, A. Saes are highly dataset dependent: A case study on the refusal direction. In *Alignment Forum*, 2024.

Kreutz-Delgado, K., Murray, J. F., Rao, B. D., Engan, K., Lee, T.-W., and Sejnowski, T. J. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2):349–396, 2003.

Leask, P., Bussmann, B., Pearce, M., Bloom, J., Tigges, C., Moubayed, N. A., Sharkey, L., and Nanda, N. Sparse autoencoders do not find canonical units of analysis. *arXiv preprint arXiv:2502.04878*, 2025.

Lehalleur, S. P., Hoogland, J., Farrugia-Roberts, M., Wei, S., Oldenziel, A. G., Wang, G., Carroll, L., and Murfet, D. You are what you eat–ai alignment requires understanding how data shapes structure and generalisation. *arXiv preprint arXiv:2502.05475*, 2025.

Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A., Shah, R., and Nanda, N. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*, 2024.

Liu, T., Puigcerver, J., and Blondel, M. Sparsity-constrained optimal transport. *arXiv preprint arXiv:2209.15466*, 2022.

Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pp. 4114–4124. PMLR, 2019.

Locatello, F., Poole, B., Ratsch, G., Scholkopf, B., Bachem, O., and Tschannen, M. Weakly-supervised disentanglement without compromises. 2020.

Lubana, E. S., Kawaguchi, K., Dick, R. P., and Tanaka, H. A percolation model of emergence: Analyzing transformers trained on a formal language. *arXiv preprint arXiv:2408.12578*, 2024.

Makhzani, A. and Frey, B. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Martins, A. and Astudillo, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pp. 1614–1623. PMLR, 2016.

Menon, A., Shrivastava, M., Krueger, D., and Lubana, E. S. Analyzing (in) abilities of saes via formal languages. *arXiv preprint arXiv:2410.11767*, 2024.

Monga, V., Li, Y., and Eldar, Y. C. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Ng, A. et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996a.

Olshausen, B. A. and Field, D. J. Wavelet-like receptive fields emerge from a network that learns sparse codes for natural images. *Nature*, 381:607–609, 1996b.

Olshausen, B. A. and Field, D. J. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Pan, W., Liu, Z., Chen, Q., Zhou, X., Yu, H., and Jia, X. The hidden dimensions of llm alignment: A multi-dimensional safety analysis, 2025. URL https://arxiv.org/abs/2502.09674.

Park, K., Choe, Y. J., and Veitch, V. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

Park, K., Choe, Y. J., Jiang, Y., and Veitch, V. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024.

Paulo, G. and Belrose, N. Sparse autoencoders trained on the same data learn different features. *arXiv preprint arXiv:2501.16615*, 2025.

Paulo, G., Shabalin, S., and Belrose, N. Transcoders beat sparse autoencoders for interpretability. *arXiv preprint arXiv:2501.18823*, 2025.

Rajamanoharan, S., Conmy, A., Smith, L., Lieberum, T., Varma, V., Kramár, J., Shah, R., and Nanda, N. Improving dictionary learning with gated sparse autoencoders. *arXiv preprint arXiv:2404.16014*, 2024a.

Rajamanoharan, S., Lieberum, T., Sonnerat, N., Conmy, A., Varma, V., Kramár, J., and Nanda, N. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024b.

Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16:1–85, 2022.

Serre, T. Learning a dictionary of shape-components in visual cortex: Comparison with neurons, humans and machines. 2006.

Simon, E. and Zou, J. Interplm: Discovering interpretable features in protein language models via sparse autoencoders. *bioRxiv*, pp. 2024–11, 2024.

Sprechmann, P. and Sapiro, G. Dictionary learning and sparse coding for unsupervised clustering. In *2010 IEEE international conference on acoustics, speech and signal processing*, pp. 2042–2045. IEEE, 2010.

Subramanian, A., Pruthi, D., Jhamtani, H., Berg-Kirkpatrick, T., and Hovy, E. Spine: Sparse interpretable neural embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Taggart, G. M. Prolu: A nonlinearity for sparse autoencoders. https://www.alignmentforum.org/posts/HEpufTdakGTTKgoYF/prolu-a-nonlinearity-for-sparse-autoencoders, 2024.

Tasissa, A., Tankala, P., Murphy, J. M., and Ba, D. K-deep simplex: Manifold learning via local dictionaries. *IEEE Transactions on Signal Processing*, 2023.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Tillmann, A. M. On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters*, 22(1):45–49, 2015. doi: 10.1109/LSP.2014.2345761.

Tolooshams, B., Matias, S., Wu, H., Temereanca, S., Uchida, N., Murthy, V. N., Masset, P., and Ba, D. Interpretable deep learning for deconvolutional analysis of neural signals. *bioRxiv*, 2024.

Von Kugelgen, J., Sharma, Y., Gresele, L., Brendel, W., Scholkopf, B., Besserve, M., and Locatello, F. Self-supervised learning with data augmentations provably isolates content from style. 2021.

Wang, Z., Liu, D., Yang, J., Han, W., and Huang, T. Deep networks for image super-resolution with sparse prior. In *Proceedings of the IEEE international conference on computer vision*, pp. 370–378, 2015.

Wu, Z., Arora, A., Geiger, A., Wang, Z., Huang, J., Jurafsky, D., Manning, C. D., and Potts, C. Axbench: Steering llms? even simple baselines outperform sparse autoencoders. *arXiv preprint arXiv:2501.17148*, 2025a.

Wu, Z., Arora, A., Wang, Z., Geiger, A., Jurafsky, D., Manning, C. D., and Potts, C. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2025b.

A. Dictionary Learning

Sparse coding (Olshausen & Field, 1996a) (alternatively known in this work as sparse dictionary learning, or just dictionary learning) was initially proposed to replicate the observed properties (”spatially localized, oriented, bandpass receptive fields”) of biological neurons in the mammalian visual cortex. It aims to invert a linear generative model with a sparsity prior on the latents:

$$\boldsymbol{x} = \boldsymbol{D}^*\boldsymbol{z}^* + \eta$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the data, $\boldsymbol{D}^* \in \mathbb{R}^{n\times s}$ is the set of s dictionary atoms, $\boldsymbol{z}^* \in \mathbb{R}^s_+$ is the sparse code, and η is additive white Gaussian noise. Given data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(P)}\}$, sparse coding performs maximum aposteriori (MAP) estimation for the dictionary $\boldsymbol{D}^*$ and representations $\boldsymbol{z}^*$ under suitably defined prior and likelihood functions (Elad et al., 2007) by solving the following optimization problem :

$$\underset{\boldsymbol{D}\in\mathcal{B},\boldsymbol{z}^{(\cdot)}\geq 0}{\arg\min} \sum_k \|\boldsymbol{x}^{(k)} - \boldsymbol{D}\boldsymbol{z}^{(k)}\|_2^2 + \lambda\mathcal{R}(\boldsymbol{z}^{(k)}) \tag{2}$$

where $\mathcal{R}(\cdot)$ is a sparsity-promoting regularizer. The set $\mathcal{B} \subseteq \mathbb{R}^{n\times s}$ includes restriction to unit norm (typical). Generally, the L1 penalty is used as the regularizer term, i.e., $\mathcal{R}(\boldsymbol{z}^{(k)}) = \|\boldsymbol{z}^{(k)}\|_1$, since using the L0 penalty makes the problem NP-hard (Tillmann, 2015). When the number of dictionary atoms is less than or equal to the dimension of input space, $s \leq n$, this is an undercomplete problem, and the sparse code can be readily obtained using the pseudo-inverse of the dictionary matrix D (provided the dictionary atoms are linearly independent), leading to the solution $\boldsymbol{z} = (\boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{D}^T\boldsymbol{x}$. Note that in this (undercomplete) case, the sparse code is a linear transformation of the input. The more interesting setting involves using an overcomplete dictionary ($s > n$), and was initially studied in (Olshausen & Field, 1997). Obtaining the sparse code $\boldsymbol{z}$ from input data $\boldsymbol{x}$ is nontrivial in this case.

In this case, sparse coding results in a sparse representation of the data and a dictionary which behaves as a data-adaptive basis. Correspondingly, sparse codes have been shown to capture interesting concepts in data (Kreutz-Delgado et al., 2003; Sprechmann & Sapiro, 2010), e.g., responding to wavelet-like regions when trained on natural images (Olshausen & Field, 1996b). In this (overcomplete) setting, a popular approach is using iterative shrinkage and thresholding algorithms (ISTA) (Daubechies et al., 2004) and their variants such as FISTA (Fast ISTA) (Beck & Teboulle, 2009). Modern approaches to this problem use ISTA to design deep residual networks with shared weights and train the network on the sparse coding objective, in a technique called Learned ISTA (LISTA) (Gregor & LeCun, 2010). Algorithm unrolling (Monga et al., 2021) is a generalization of this technique and involves designing *interpretable* neural networks using iterative algorithms where each layer of the network reflects an iteration of the algorithm. These networks are interpretable since the weights correspond to an underlying process which was used to design the iterative algorithm. Unrolling has widespread applications in signal processing, and is extensively reviewed in (Monga et al., 2021).

We also note that sparse coding has been used with algorithm unrolling as a model-based interpretable deep learning technique for a wide range of applications, including image super-resolution (Wang et al., 2015), graph signal denoising (Chen et al., 2021), mechanical fault diagnosis (An et al., 2022), deconvolving neural activity of dopamine neurons in mice (Tolooshams et al., 2024). Therefore, assuming a linear generative model of data (Eq. 2) where the dictionary atoms are physically relevant in some application, sparse coding using an unrolled network learns the underlying interpretable dictionary atoms.

B. Related Work

SAEs (Ng et al., 2011) approximate sparse dictionary learning by using a single hidden layer to compute the sparse code from data. For input $\boldsymbol{x} \in \mathbb{R}^d$,

$$\text{(i) } \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}^\top\boldsymbol{x} + \boldsymbol{b}_e), \quad \text{and} \quad \text{(ii) } \hat{\boldsymbol{x}} = \boldsymbol{D}\boldsymbol{z} + \boldsymbol{b}_d, \tag{3}$$

SAEs are a specific instantiation of the broader agenda of dictionary learning tools for concept-level explainability (Kim et al., 2018; Olah et al., 2020; Fel, 2025; Faruqui et al., 2015; Subramanian et al., 2018; Arora et al., 2018). A number of SAE architectures have been proposed recently, including ReLU SAE (Bricken et al., 2023), TopK SAE (Gao et al., 2024; Makhzani & Frey, 2013), gated SAE (Rajamanoharan et al., 2024a), JumpReLU SAE (Rajamanoharan et al., 2024b), Batch TopK SAE ((Bussmann et al., 2024)), ProLU SAE ((Taggart, 2024)), and so on. While promising results have been discovered, e.g., latents that respond to concepts of refusal, gender, text script (Bricken et al., 2023; Templeton et al., 2024; Durmus et al., 2024), foreground vs. background concepts (Fel et al., 2023), and concepts of protein structures (Simon & Zou, 2024; Garcia & Ansuini, 2025; Adams et al., 2025), a series of negative results have started to emerge on the limitations of SAEs. For example, (Bhalla et al., 2024; Wu et al., 2025a) show a mere prompting baseline can outperform model control compared to SAE or probing based feature ablation baseline. Similar results were observed by (Menon et al., 2024) in a narrower formal language setting. Meanwhile, criticizing the underlying linear representation hypothesis that has informed design of earlier SAE architectures (specifically, the vanilla ReLU SAEs), (Engels et al., 2024a;b) has shown that SAE features can be multidimensional and nonlinear. Importantly, recent results from (Fel et al., 2025; Paulo & Belrose, 2025; Kissane et al., 2024) have shown that two SAEs trained on the exact same data, just with a different seed, can yield very different concepts and hence very different interpretations. These results are related to the lack of canonical nature in SAE latents (Leask et al., 2025) This behavior, often called algorithmic instability, makes reliability of SAEs challenging for any practical purposes. More broadly, given the hefty research investment going into the topic, we believe it is warranted that a more formal and theoretical account help solidify the limitations and challenges SAEs (or at least the current paradigm thereof) faces. This can help steer the research in a direction that yields meaningful improvement in SAEs, e.g., in their practical utility. This motivation underscores our work. For a related effort on this front, we highlight the work by (Ayonrinde et al., 2024), who contextualize SAEs from a minimum-description length perspective and enable an intuitively solid account of how features may split to overly specialized concepts (e.g., tokens).

Disentangled Representation Learning. As mentioned in Sec. E, results similar to ours have been reported in the field of disentangled representation learning, wherein one aims to invert a data-generating process to identify the factors of variants (i.e., latent variables) that underlie it. To this end, autoencoders were used as a popular tool, since they offer a method that can (ideally) simultaneously invert the generative process and identify the underlying latents (Higgins et al., 2017). However, (Locatello et al., 2019) showed that in fact this problem is rather challenging: unless one designs an autoencoder architecture that bakes-in assumptions about the generative process, i.e., the precise function mapping itself, there are no guarantees the retrieved latents will correspond to the ground-truth ones. This result led to design of several methods focused on exploiting "weak supervision", i.e., extra information available from data-pairs such as multiple views of an image or temporally consistent video frames, to circumvent the theoretical challenges of disentanglement (Locatello et al., 2020; Gresele et al., 2020; Von Kugelgen et al., 2021). Our contributions are similar in nature to these results on disentanglement, but we (i) specifically focus on the context of SAEs and (ii) provide a more concrete proof that establishes precisely what the inductive biases of popular SAEs are, i.e., what concepts the SAEs are biased towards uncovering. Having established these results, we now believe the next step that the disentanglement community took, i.e., use of weak supervision, would make sense for the SAEs community as well. This can involve exploiting temporal correlations between tokens in a sentence, or the fact that representations across layers do not change much, as in Crosscoders and Transcoders (Lindsey et al., 2024; Dunefsky et al., 2025; Paulo et al., 2025).

C. Experimental Setup

The synthetic experiments (separability, heterogeneity) and vision experiments were run on NVIDIA A100 40GB GPUs, while the formal language experiments were run on NVIDIA RTX A6000 48GB GPUs.

C.1. Separability experiment

We construct a synthetic dataset consisting of six isotropic Gaussian clusters in a two-dimensional (2D) space. The cluster centers are arranged such that adjacent clusters are separated by an angular difference of $2\pi/6$, with alternate clusters having norms of 1 and 3. Each cluster is sampled from a multivariate normal distribution with a variance of $2^{-5.5}$. The dataset consists of 1 million data points per concept, yielding a total of 6 million samples. Of these, we use 70% (700,000 points) for training.

Our experiments evaluate four sparse autoencoder (SAE) architectures: ReLU SAE, JumpReLU SAE, TopK SAE, and SpaDE. The first three architectures are implemented following their original formulations (in (Bricken et al., 2023),(Rajamanoharan et al., 2024b),(Gao et al., 2024)), with the decoder activations normalized in the forward pass. The SpaDE model follows the same single hidden-layer autoencoder structure but differs in that it utilizes Euclidean distance computations and a SparseMax activation function for the encoder. Across all models, the hidden-layer width is set to 128, and a pre-encoder bias is used in all cases except for SpaDE.

For training, the (inverse) temperature parameter λ in SpaDE is initialized to $1/(2 \times \text{input dimension})$ and parameterized using the Softplus function to ensure non-negativity. This parameter trained along with the encoder and decoder weights, to allow the model to *learn* its desired sparsity level. Note that large values of λ lead to greater sparsity since Sparsemax is scale-sensitive. In JumpReLU, the threshold is initialized at 10^{-3} across all latent dimensions, with a bandwidth of 10^{-3} for the straight-through estimator (STE), as it is proposed in (Rajamanoharan et al., 2024b). All models are trained using the Adam optimizer with a learning rate of 10^{-2}, which follows a cosine decay schedule from 10^{-2} to 10^{-4}. The momentum parameter is set to 0.9, and we use a batch size of 512. Training runs for approximately 8000 iterations, and gradient clipping is applied (gradient norms are clipped at 1) to stabilize optimization.

Regularization parameters are selected such that sparsity levels remain comparable across models. Specifically, the regularization coefficient γ is chosen in the range 10^{-6} to 1 for ReLU and JumpReLU SAEs, between 4 and 64 (powers of 2) for TopK SAE, and in the range 10^{-6} to 1 for SpaDE. Each model applies a different regularization strategy: ReLU SAE uses L_1 regularization, JumpReLU SAE applies L_0 regularization with a straight-through estimator (STE) as in (Rajamanoharan et al., 2024b), TopK SAE does not use explicit regularization but incorporates an auxiliary loss term as in (Gao et al., 2024), with $K_{aux} = k$ (same as the choice of sparsity level k in TopK) with $\gamma_{\text{aux}} = 1$ (the scaling for the auxiliary loss term), and SpaDE employs a distance-weighted L_1 regularization, which comes from (Tasissa et al., 2023).

All networks are initialized such that the decoder weights are initially set as the transpose of the encoder weights, though they are allowed to update freely during training. Model weights are sampled from a normal distribution $\mathcal{N}(0, 1)$. To maintain consistency in scale between inputs and latent activations, a scaling factor λ is applied to all latent units, given by $\lambda \approx 1/2 \times$ input dimension (note that this is not trainable for ReLU, JumpReLU and TopK SAEs). Across all architectures, we use the Mean Squared Error (MSE) loss function, with the regularizers and regularizer scaling constants as described above.

For evaluation, we analyze a subset of 1000 data points per concept. The primary metric for comparison is the F1-score, which is computed based on precision and recall. Precision is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \tag{4}$$

while recall is given by:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}. \tag{5}$$

Using these definitions, the F1-score is computed as:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{6}$$

In our setup, precision and recall are computed by thresholding latent activations at 10^{-6}. Additionally, we analyze the receptive fields by creating a 2D meshgrid, passing all points through the model, and extracting their SAE latent representations. Cosine similarities between pairs of data points are also computed by obtaining their latent representations, calculating the pairwise cosine similarity, and organizing the results by class.

To further examine latent space structure, we compute the stable rank of the representation matrix. Stable rank for the similarity matrix is computed as the sum of singular values divided by the largest singular value (alternatively called the intrinsic dimension of this matrix):

$$\text{Stable Rank} = \frac{\sum \sigma_i}{\sigma_{\max}}. \tag{7}$$

Finally, spectral clustering is performed on the similarity matrix derived from latent representations. The number of clusters is determined by the stable rank of this similarity matrix (rounded up), providing insights into the correlations between SAE latent representations.

C.2. Heterogeneity experiment

We construct a synthetic dataset consisting of five isotropic Gaussian clusters in a 128-dimensional space. The intrinsic dimensionality of each cluster follows the sequence $2^q - 2$ for different values of $q \in \{3, 4, 5, 6, 7\}$, resulting in clusters with intrinsic dimensions of $6, 14, 30, 62, 126$, respectively. The lower-dimensional clusters belong to subspaces that form strict subsets of the subspaces of higher-dimensional ones, meaning that the first six dimensions are fully contained in the next 14, which are further contained in the next 30, and so on up to 126 dimensions. Cluster centers are sampled uniformly at random from the range $[0, \frac{1}{21}]$ along each dimension. The variance of each concept is chosen to be inversely proportional to its intrinsic dimension to ensure that the total variance per concept remains constant across all concepts. The dataset contains 6.4 million data points per concept, yielding a total of 32 million samples, of which 70% (approximately 22 million points) are used for training.

Our models follow four different sparse autoencoder (SAE) architectures: ReLU SAE, JumpReLU SAE, TopK SAE, and SpaDE. The first three are implemented according to their original formulations in (Bricken et al., 2023), (Rajamanoharan et al., 2024b), and (Gao et al., 2024), with the decoder activations normalized in the forward pass. The SpaDE model follows the same single hidden-layer autoencoder structure but differs in that it utilizes Euclidean distance computations and a SparseMax activation function for the encoder. Across all models, the SAE hidden-layer width is set to 512. A pre-encoder bias is applied in all cases except for SpaDE. Additionally, for the TopK SAE, a ReLU activation is applied before selecting the top k latent dimensions.

For training, the temperature parameter λ in SpaDE is initialized at $1/(2 \times \text{input dimension})$ and parameterized using the Softplus function to ensure non-negativity. This parameter trained along with the encoder and decoder weights, to allow the model to *learn* its desired sparsity level. In JumpReLU, the threshold is initialized at 10^{-3} across all latent dimensions, with a bandwidth of 10^{-3} for the straight-through estimator (STE). All models are trained using the Adam optimizer with a learning rate of 10^{-2}, which follows a cosine decay schedule from 10^{-2} to 10^{-4}. The momentum parameter is set to 0.9, and we use a batch size of 2048. Training runs for approximately 10,000 iterations, and gradient clipping (restricting gradient norms to be less than 1) is applied to stabilize optimization.

Regularization parameters are selected such that sparsity levels remain comparable across models. Specifically, the regularization coefficient γ is chosen in the range 10^{-3} to 5.0 for ReLU SAE, 10^{-3} to 1 for JumpReLU SAE, from 4 to 256 (powers of 2) for TopK SAE, and from 10^{-3} to 10 for SpaDE. Each model applies a different regularization strategy: ReLU SAE uses L_1 regularization, JumpReLU SAE applies L_0 regularization with a straight-through estimator (STE) following from (Rajamanoharan et al., 2024b), TopK SAE does not use explicit regularization but incorporates an auxiliary loss term with $\gamma_{\text{aux}} = 1$ (scaling for the auxillary term in the loss) and $K_{aux} = k$ (same as sparsity level), and SpaDE employs a distance-weighted L_1 regularization.

All networks are initialized such that the decoder weights are initially set as the transpose of the encoder weights, though they are allowed to update freely during training. Model weights are sampled from a normal distribution $\mathcal{N}(0, 1)$. To maintain consistency in scale between inputs and latent activations, a scaling factor λ is applied to all latent units, given by $\lambda \approx 1/2 \times$ input dimension. Across all architectures, we use the Mean Squared Error (MSE) loss function.

For evaluation, we analyze a subset of 1000 data points per concept. We report the *normalized MSE*, defined as the ratio of the standard MSE to the variance of the corresponding concept:

$$\text{Normalized MSE} = \frac{\text{MSE}}{\text{Variance of Concept}}. \tag{8}$$

We also compute *sparsity* (L_0) per concept, measured as the average number of active latents per data point, averaged over each concept.

To analyze latent representations, we examine cosine similarities in two contexts: (i) between pairs of SAE latent representations for different input data points (per-input co-occurrence) and (ii) between pairs of latents aggregated over all data points (global co-occurrence). For the latter, each latent is assigned a *concept label* based on the concept for which it is most frequently activated on average. This assignment provides insight into how latents specialize across different underlying structures in the dataset.

C.3. Formal Languages experiment

Data. The formal language setup analyzed in the main paper (Sec. F.3) involves training a 2-layer nanoGPT model on strings from an English-like PCFG. Broadly, a PCFG is defined via a 5-tuple $G = (\mathtt{NT}, \mathtt{T}, \mathtt{R}, \mathtt{S}, \mathtt{P})$, where $\mathtt{NT}$ is a finite set of non-terminal symbols; $\mathtt{T}$ is a finite set of terminal symbols, disjoint from $\mathtt{NT}$; $\mathtt{R}$ is a finite set of production rules, each of the form $A \to \alpha\beta$, where $A \in \mathtt{NT}$ and $\alpha, \beta \in (\mathtt{NT} \cup \mathtt{T})$; $\mathtt{S} \in \mathtt{NT}$ is the start symbol; and $\mathtt{P}$ is a function $\mathtt{P} : \mathtt{R} \to [0, 1]$, such that for each $A \in \mathtt{NT}$, $\sum_{\alpha : A \to \alpha \in \mathtt{R}} \mathtt{P}(A \to \alpha\beta) = 1$. To *generate* a sentence from the grammar, the following process is used.

1. Start with a string consisting of the start symbol S.
2. While the string contains non-terminal symbols, randomly select a non-terminal A from the string. Choose a production rule $A \to \alpha\beta$ from $\mathtt{R}$ according to the probability distribution $\mathtt{P}(A \to \alpha)$.
3. Replace the chosen non-terminal A in the string with α, the right-hand side of the production rule.
4. Repeat the production rule selection and expansion steps until the string contains only terminal symbols (i.e., no non-terminals remain).
5. The resulting string, consisting entirely of terminal symbols, is a sentence sampled from the grammar.

We follow the same rules of the grammar considered in (Menon et al., 2024). The strings are tokenized via one-hot encoding via a manually defined tokenizer.

Model training. Models are trained from scratch on strings sampled from the grammar above. Strings are padded to length 128 (if not already that length), and a batch-size of 128 ($\sim$10K tokens per batch) is used for training. Training uses Adam optimizer with a cosine learning-rate schedule starting at 10^{-3} and ending at 10^{-4} after 70K iterations, alongside a weight decay of 10^{-4}. The nanoGPT models used in this work have a width of 128 units, with an MLP expansion factor of 2 and also 2 attention heads per attention layer.

SAE training. All SAEs trained in the formal language setup involve an expansion factor of 2×, i.e., 256 latents for a residual stream of 128 dimensions. Training involves a constant learning rate of 10^{-3} and lasts for 10K iterations ($\sim$1M tokens). We sweep regularization strength for SAEs' training, yielding SAEs with different sparsity levels. While we fix the regularization strength for SpaDE based on best values identified from the synthetic, Gaussian cluster datasets, for other SAEs (ReLU, JumpReLU, and TopK) we report the best possible results from our sweep by looking at the top-10 per-concept F1 scores; i.e., reported results are a best-case estimate of results achievable by training of these SAEs, and in practice performance can be expected to be poorer than what we analyze. Cross-task transfer for SpaDE's hyperparameters is intriguing in this regard, since we found other SAEs' hyperparameters to not transfer.

C.4. Vision experiment

Data. We use an off-the-shelf, large-scale pretrained model for our analysis in these experiments, specifically *DINOv2-base* (with registers). For simplicity, we focus on a 10-class subset of ImageNet, called *Imagenette*, containing 1.5k images per class. Representations are extracted from the model for images of these classes, yielding 261 tokens per image.

SAE training. SAEs are trained on all available tokens, including spatial, CLS, and registers tokens, for 50 epochs with 200 latent dimensions. With 261 tokens per image, this amounts to $\sim$200M tokens for training SAEs over the course of 50 training epochs. For each SAE, the best reconstruction is selected based on a sparsity-controlled learning rate sweep. This resulted in an optimal learning rate of 5×10^{-4} for TopK, ReLU, and SpaDE, while JumpReLU performed best with 10^{-4} (using Adam optimizer). Additionally, we note our JumpReLU implementation employs a Silverman kernel with a bandwidth of 10^{-2}, which we found to work best for our setting.

D. Further Theory Results

E. Unified Framework for SAEs

In this section, we develop a framework which captures multiple SAEs used in practice. More specifically, we analyze the following three popular SAE architectures: ReLU SAE (Cunningham et al., 2023; Bricken et al., 2023), TopK SAE (Gao et al., 2024; Makhzani & Frey, 2013) and JumpReLU

Table 3: **Projection Nonlinearities in SAE Encoders.** Each model can be understood by its nonlinear orthogonal projection $\boldsymbol{g}(\cdot)$ onto a constraint set $\mathcal{S}$ which determines its activation behavior, sparsity structure, and implicit data assumptions.

Model	$\boldsymbol{g}(\boldsymbol{v})$
ReLU	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\}, \mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : \boldsymbol{x} \geq 0\}$
TopK	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\}, \mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : \boldsymbol{x} \geq \mathbf{0}, \|\|\boldsymbol{x}\|\|_0 \leq k\}$
Heaviside (H)	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v} + \frac{1}{2}\mathbf{1}\right\}, \mathcal{S} = \{0, 1\}^s$
JumpReLU	$\mathrm{ReLU}(\boldsymbol{v} - \boldsymbol{\theta}) + \boldsymbol{\theta} \odot H(\boldsymbol{v} - \boldsymbol{\theta})$

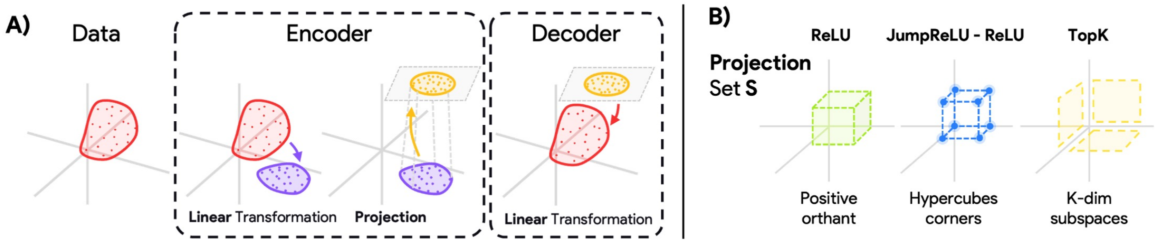

Figure E.1: **Projection As The Key Architectural Difference Between SAEs. A)** SAE encoders do more than just linearly transform data—they project it onto an architecture-specific constraint set. This projection fundamentally determines which features an SAE can extract and which it will suppress. **B)** Different SAEs rely on different projection sets $\mathcal{S}$: ReLU projects onto the positive orthant, TopK onto $K-$sparse subspaces, and JumpReLU combines ReLU with a projection onto a hypercube (via a Heaviside step function).

SAE (Rajamanoharan et al., 2024b; Lieberum et al., 2024). This framework unravels a duality between how concepts are encoded in model representations and an SAE's architecture. The nonlinearity of the SAEs under study is an orthogonal projection onto some set, where the choice of projection set differentiates SAEs (see Fig. E.1). We formalize such nonlinearities as projection nonlinearities, as defined below.

Definition E.1 (Projection Nonlinearity)**.** Let $\boldsymbol{v} \in \mathbb{R}^s$ be a pre-activation vector. A projection nonlinearity $\boldsymbol{\Pi}_{\mathcal{S}}\left\{\cdot\right\} : \mathbb{R}^s \to \mathbb{R}^s$ is defined as:

$$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\} = \underset{\boldsymbol{\pi}\in\mathcal{S}}{\arg\min}\, \|\boldsymbol{\pi} - \boldsymbol{v}\|_2^2, \tag{9}$$

where $\mathcal{S} \subseteq \mathbb{R}^s$ is the constraint set onto which $\boldsymbol{v}$ is orthogonally projected. Popular SAE nonlinearities, e.g., ReLU, JumpReLU, and TopK, are orthogonal projections onto different sets (see Tab. 3).

Generalizing the variational form of projection nonlinearities allows us to formalize SAEs as follows.

Claim E.1 (Bilevel optimization of SAEs)**.** *A sparse autoencoder (Eq. 3) with the dictionary learning loss function (Eq. 2) solves the following bi-level optimization problem:*

$$\begin{aligned}&\underset{\boldsymbol{D}\in\mathcal{B},\boldsymbol{z}\geq\boldsymbol{0}}{\arg\min}\ \sum_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2 + \lambda\mathcal{R}(\boldsymbol{z})\\ &\quad s.t.\ \ \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) \in \underset{\boldsymbol{\pi}\in\mathcal{S}}{\arg\min}\, \boldsymbol{F}(\boldsymbol{\pi}, \boldsymbol{W}, \boldsymbol{x}),\end{aligned} \tag{10}$$

where $\boldsymbol{F}$ is a variational formulation of the SAE encoder $\boldsymbol{f}$. For SAEs, $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}^\top\boldsymbol{x} + \boldsymbol{b}_e)$ (Eq. 3). Note that this inner optimization with the objective $\boldsymbol{F}$ is what differentiates different SAEs.

Proof. The outer optimization follows from the dictionary learning loss with sparsity-inducing penalty of the SAE (Eq. 2). The constraint is imposed by the SAE encoder's architecture (Eq. 3). The variational formulation of the encoder as the minimization of some objective $\boldsymbol{F}$ over set $\mathcal{S}$ is a generalization of projection nonlinearities (Eq. 9) for which $\boldsymbol{F}(\boldsymbol{\pi}, \boldsymbol{W}, \boldsymbol{x}) = \|\boldsymbol{W}^\top\boldsymbol{x} + \boldsymbol{b}_e - \boldsymbol{\pi}\|_2^2$. □

This framework implies that each SAE solves a different, constrained (through encoder architecture) optimization version of sparse dictionary learning. *This constraint dictates the quality of the solution obtained*, since it restricts the search space of solutions to dictionary learning, and hence does not have to capture the full sparse coding solution. To further formalize this claim in the next section, we now define receptive fields, a popularly used concept in neuroscience to study the response properties of biological neurons (Olshausen & Field, 1997).

Definition E.2 (Receptive Field)**.** Consider a neuron k, which computes a function $f^{(k)} : \mathbb{R}^d \to \mathbb{R}$. The receptive field of this neuron is defined as $\mathcal{F}_k = \{\boldsymbol{x} \in \mathbb{R}^d \mid f^{(k)}(\boldsymbol{x}) > 0\}$.

Intuitively, $\mathcal{F}_k$ represents the region of input space where neuron k is active. *The structure of receptive fields in an SAE is dictated by its encoder's architecture.*

Duality: Properties of the SAE encoder will constrain receptive fields' structure for SAE latents. These constraints directly translate to assumptions (often *implicit*, see Sec. 3) about the data structure, since "monosemanticity" (Bricken et al., 2023; Elhage et al., 2022) requires receptive fields to match structure of concepts in data. Alternatively, if one knows how concepts are organized in the data (model representations), duality can be used to design an appropriate SAE architecture (see Sec. 3.1).

Fundamental Limitation of SAEs

An SAE's encoder enforces *implicit dual assumptions about data*, fundamentally shaping which concepts it can identify and which remain obscure. To build more effective SAEs, these assumptions must *explicitly match the true structure of the data.*

E.1. Projections and Nonlinearities

The nonlinearity of popular SAEs is commonly an orthogonal projection onto some set, where the choice of projection set differentiates SAEs (see Fig. E.1). We formalize such nonlinearities as projection nonlinearities, as (re)defined below.

Definition E.3 (Projection Nonlinearity)**.** Let $\boldsymbol{v} \in \mathbb{R}^s$ be a pre-activation vector. A projection nonlinearity $\boldsymbol{\Pi}_{\mathcal{S}}\left\{\cdot\right\} : \mathbb{R}^s \to \mathbb{R}^s$ is defined as:

$$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\} = \arg\min_{\boldsymbol{\pi}\in\mathcal{S}} \|\boldsymbol{\pi} - \boldsymbol{v}\|_2^2, \tag{11}$$

where $\mathcal{S} \subseteq \mathbb{R}^s$ is the constraint set onto which $\boldsymbol{v}$ is orthogonally projected. The structure of $\mathcal{S}$ determines the properties of the nonlinearity.

We will say a function $\boldsymbol{f}(\cdot)$ is a **Projection Encoder** if it uses a projection nonlinearity $\boldsymbol{g}(\cdot)$ applied to a linear transformation of the input. This is equivalent to using $\boldsymbol{v} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}_e$, and $\boldsymbol{f} = \boldsymbol{g}(\boldsymbol{v})$ (see Eq. 3), where $\boldsymbol{g}$ is a projection nonlinearity. Popular SAEs can be understood as a similar Projection Encoder with different projection nonlinearities, as shown in Tab. 4 (see Theorem E.5 for a derivation).

Table 4: **Projection Nonlinearities in SAE Encoders.** Each model can be understood by its nonlinear orthogonal projection $\boldsymbol{g}(\cdot)$ onto a constraint set $\mathcal{S}$ which determines its activation behavior, sparsity structure, and implicit data assumptions.

Model	$\boldsymbol{g}(\boldsymbol{v})$
ReLU	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\}, \mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : \boldsymbol{x} \geq 0\}$
TopK	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v}\right\}, \mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : \boldsymbol{x} \geq \boldsymbol{0}, \|\boldsymbol{x}\|_0 \leq k\}$
Heaviside (H)	$\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{v} + \frac{1}{2}\boldsymbol{1}\right\}, \mathcal{S} = \{0,1\}^s$
JumpReLU	ReLU$(\boldsymbol{v} - \boldsymbol{\theta}) + \boldsymbol{\theta} \odot H(\boldsymbol{v} - \boldsymbol{\theta})$

Lemma E.4 (Elementwise projections)**.** *For projection nonlinearities whose projection sets satisfy componentwise constraints, i.e.* $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : f(x_j) \leq 0, h(x_k) = 0 \forall j,k \in [s]\}$*, the projection problem can be decoupled and broken down into a combination of elementwise projections, leading to an elementwise nonlinearity. The converse is also true: any elementwise nonlinearity which is also a projection nonlinearity can be written as a combination of elementwise projections, leading to componentwise constraints on the projection set*

Proof.

$$\begin{aligned}
\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\} &= \arg\min_{\boldsymbol{\pi}\in\mathcal{S}} \|\boldsymbol{\pi} - \boldsymbol{x}\|^2 && (12)\\
&= \arg\min_{f(\pi_j)\leq 0, g(\pi_j)=0, j\in[s]} \sum_k (\pi_k - x_k)^2 && (13)\\
&= (\ldots, \arg\min_{f(\pi_k)\leq 0, g(\pi_k)=0,} (\pi_k - x_k)^2, \ldots) && (14)\\
\text{i.e.,}\quad \Pi_{\mathcal{S}}\{\boldsymbol{x}\}_k &= \arg\min_{f(\pi_k)\leq 0, g(\pi_k)=0,} (\pi_k - x_k)^2 && (15)
\end{aligned}$$

This is a consequence of the objective function above (squared euclidean norm of the difference $\boldsymbol{\pi} - \boldsymbol{x}$) decomposing into a sum over componentwise functions. The above argument can be traced backward, since all steps are invertible, which proves the converse. □

Theorem E.5 (Projection Nonlinearities)**.** *ReLU, TopK, JumpReLU are simple combinations of orthogonal projections onto nonlinearity-specific sets: ReLU is a projection onto the positive orthant, TopK is a projection onto the union of all k-sparse subspaces, and JumpReLU is a sum of shifted ReLU and shifted Heaviside step, which itself is a projection onto the corners of a hypercube.*

Proof. First consider the ReLU nonlinearity, defined for $\boldsymbol{x} \in \mathbb{R}^s$ as:

$$\boldsymbol{z} = \text{ReLU}(\boldsymbol{x}) \tag{16}$$

$$z_i = \begin{cases} x_i \;\text{ if }\; x_i \geq 0 \\ 0 \text{ else} \end{cases} \tag{17}$$

This is an elementwise nonlinearity, so it suffices to show that each component can be written as a projection (from Lemma E.4). Consider this reformulation:

$$z_i = \arg\min_{\pi_i \geq 0} (x_i - \pi_i)^2 \tag{18}$$

This is equivalent to ReLU, since for all non-negative values, it equals the input, while it is 0 (nearest non-negative point) for all negative inputs. Using Lemma E.4, ReLU is a projection nonlinearity with projection set $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : x_i \geq 0 \forall i \in [s]\}$.

JumpReLU is defined as:

$$\begin{aligned} \text{JumpReLU}(\boldsymbol{x}) &= \boldsymbol{x} \odot \mathbb{H}(\boldsymbol{x} - \boldsymbol{\theta}) && (19) \\ &= (\boldsymbol{x} - \boldsymbol{\theta} + \boldsymbol{\theta}) \odot \mathbb{H}(x - \theta) && (20) \\ &= \text{ReLU}(\boldsymbol{x} - \boldsymbol{\theta}) + \boldsymbol{\theta} \odot \mathbb{H}(\boldsymbol{x} - \boldsymbol{\theta}) && (21) \end{aligned}$$

where the heaviside step function $\mathbb{H}$ is:

$$\mathbb{H}(\boldsymbol{x}) = \mathbb{I}(\boldsymbol{x} > 0) \tag{22}$$

which is performed elementwise. Thus, JumpReLU (and the heaviside step) is also an elementwise nonlinearity. Consider the step function:

$$\begin{aligned} \mathbb{H}(\boldsymbol{x})_i = \mathbb{H}(x_i) &= \begin{cases} 1 \;\text{ if } x_i \geq 0 \\ 0 \;\text{ else} \end{cases} && (23) \\ &= \arg\min_{\pi_i \in \{0,1\}} (x_i + 0.5 - \pi_i)^2 && (24) \end{aligned}$$

which is a shifted version of a projection. Again using Lemma E.4, $\mathbb{H}$ is a projection nonlinearity with projection set $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^s : x_i \in \{0, 1\}\}$, i.e., the corners of a unit hypercube.

The TopK nonlinearity is defined as:

$$\begin{aligned} y_j &= \text{ReLU}(x_j) && (25) \\ \text{TopK}(\boldsymbol{x})_j &= y_j \; \mathbb{I}\big(y_j \geq y_p \forall p \in \mathcal{M} : |\mathcal{M}| = s - K\big) && (26) \end{aligned}$$

where s is the dimension of the space. Note that topK typically includes a ReLU applied first ((Gao et al., 2024)), making all entries of the vector non-negative followed by choosing the k-largest entries of $ReLU(\boldsymbol{x})$. Consider a projection onto the union of all k-dimensional axis-aligned subspaces. With non-negative entries (due to ReLU), this would lead to choosing the k largest entries of x:

$$\begin{aligned} \arg\min_{\pi:\, \pi \text{ is } k-\text{sparse}} \|x - \pi\|_2^2 &= \arg\min_{\boldsymbol{\pi}:\, \boldsymbol{\pi} \text{ is } k-\text{sparse}} \sum_i (x_i - \pi_i)^2 && (27) \\ &= TopK(\boldsymbol{x}) && (28) \end{aligned}$$

This completes the proof. □

Theorem E.6. *Projection nonlinearities satisfy the following properties:*

1. *For points within the set $\mathcal{S}$, projection is an identity map*

$$\boldsymbol{x} \in \mathcal{S} \implies \boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\} = \boldsymbol{x}$$

2. *For points outside the set $\mathcal{S}$, projection is onto the boundary*

$$\boldsymbol{x} \notin \mathcal{S} \implies \boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\} \in \partial\mathcal{S}$$

3. *If $\partial\mathcal{S}$ is a flat (linear manifold), or a subset of a flat (with flat boundaries), projection of points outside the set $\mathcal{S}$ is either piecewise linear or constant:*

$$\begin{aligned} \boldsymbol{\Pi}_{\mathcal{S}}\left\{\alpha\boldsymbol{x}_1 + \beta\boldsymbol{x}_2\right\} &= \alpha\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}_1\right\} + \beta\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}_2\right\} \;\; \textit{for } \alpha, \beta \in \mathcal{T}, \; \textit{OR} \\ \boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\} &= \boldsymbol{c}, \; \boldsymbol{x} \in \boldsymbol{D} \; \textit{(a linear piece)} \end{aligned}$$

where $\boldsymbol{x}_1, \boldsymbol{x}_2 \notin \mathcal{S}$, $\mathcal{T} \subseteq \mathbb{R}$ is suitably defined to confine $\boldsymbol{x}$ to the corresponding linear piece

Proof. (sketch) (1) is trivial and follows from the definition of projection nonlinearities (Eq. 9).
For (2), suppose $\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\}$ is in the interior of $\mathcal{S}$. This implies that $\exists \boldsymbol{y} \in Int(\mathcal{S})$ such that $\boldsymbol{y} = \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\}, \alpha \in (0,1]$ and therefore $\|\boldsymbol{y} - \boldsymbol{x}\|^2 < \|\boldsymbol{x} - \boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\}\|^2$, which is a contradiction. Thus $\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\} \in \partial\mathcal{S}$.
For (3), one can consider the section of the boundary $\partial\mathcal{S}$ that is closest to $\boldsymbol{x}$, and extend it to form a subspace (possible since it is flat). Since projections onto subspaces are linear operations, $\boldsymbol{\Pi}_{\mathcal{S}}\left\{\boldsymbol{x}\right\}$ is linear in some neighborhood, and thus piecewise linear. In some cases, there is a single *corner* point of $\mathcal{S}$ that is closest to $\boldsymbol{x}$, in which case the projection is a constant. □

Projection nonlinearities are orthogonal projections onto various sets. For points within the set $\mathcal{S}$, projection is the point itself, while for points outside, the projection is onto the boundary $\partial\mathcal{S}$ (Theorem E.6 in Appendix). For projections to be well defined everywhere, the set $\mathcal{S}$ must be closed (so that the boundary belongs to the set, i.e., $\partial\mathcal{S} \in \mathcal{S}$). Note that if the set $\mathcal{S}$ is a subspace of $\mathbb{R}^s$, projection is a linear map. Therefore, the nonlinearity of projection nonlinearities comes from choosing either a subset of a subspace, or a non-flat manifold. Sparsity in projection nonlinearities is a consequence of the projection set having edges/corners along sparse subspaces.

E.2. Receptive fields of various SAEs

First, we (re)define the four SAE encoders we study in this section:

$$\text{ReLU SAE: } \boldsymbol{z} = \text{ReLU}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}) \tag{29}$$

$$\text{JumpReLU SAE: } \boldsymbol{z} = \text{JumpReLU}(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}) \tag{30}$$

$$\text{TopK SAE: } \boldsymbol{z} = \text{TopK}(\boldsymbol{W}^T\boldsymbol{x}) \tag{31}$$

$$\text{SpaDE: } \boldsymbol{z} = \text{Sparsemax}(-\lambda d(\boldsymbol{x}, \boldsymbol{W})) \tag{32}$$

$$d(\boldsymbol{x}, \boldsymbol{W})_i = \|\boldsymbol{x} - \boldsymbol{w}_i\|_2^2 \tag{33}$$

This section discusses the piecewise linear (affine) regions (by showing that each of the above is a piecewise linear function) and neuron receptive fields in input space for each of the four SAEs (ReLU, JumpReLU, TopK, SpaDE). Projection nonlinearities become piecewise linear when the projection sets have flat faces. Under the requirement of monosemanticity, the structure of receptive fields directly implies the assumption that concepts in data have the same structure as the receptive field.

For projection-based encoders, the receptive field can be rewritten as

$$\mathcal{F}_k = \boldsymbol{f}^{-1}\big(\mathcal{S} \cap \{z_k > 0\}\big),$$

where $\mathcal{S}$ is the projection set of the encoder.

That is, $\mathcal{F}_k$ is the pre-image of the intersection of the projection set with the half-space $\{z_k > 0\}$. Alternatively, it can be viewed as the complement of the pre-image of the set $\mathcal{S} \cap \{z_k = 0\}$, where the hyperplane $z_k = 0$ indicates latent k is "dead". This expression shows the explicit relation between the projection set and the receptive field properties of the SAE.

First note that all four nonlinearities have some level of sparsity, i.e., some neurons are *turned off* at times. The following observation is crucial in formulating the piecewise linear regions:

Lemma E.7 (Gating)**.** *Given the indices $\mathcal{M} = \{i_1, i_2, ..., i_{|\mathcal{M}|}\}$ of active neurons (with nonzero outputs), ReLU, JumpReLU, TopK and Sparsemax are all affine functions of their inputs.*

Lemma E.7 indicates that the nonlinearity in these transformation lies only in their *gating*, or selection of active indices. Thus, each linear (affine) region is characterized by a specific choice of indices $\mathcal{M}$ of active neurons. Note that not all choices of indices may be allowed by the nonlinearity. Denote the set of allowed indices by $\mathbb{M}$.
Let $\mathcal{L}_\mathcal{M} \subseteq \mathbb{R}^n$ denote the piecewise linear (affine) region corresponding to active indices $\mathcal{M}$.

Lemma E.8. *The set $\{\mathcal{L}_\mathcal{M} : \mathcal{M} \in \mathbb{M}\}$ of all piecewise linear regions forms a partition of $\mathbb{R}^n$.*

Using the Gating lemma, we can associate each set of active indices to a piecewise linear region, and identify receptive fields as unions of such piecewise linear regions.

Lemma E.9 (Receptive fields and piecewise linear regions)**.** *A neuron's receptive field is a union of piecewise linear regions where the neuron is active:*

$$\mathcal{F}_k = \cup_{\mathcal{M}:k\in\mathcal{M}} \mathcal{L}_\mathcal{M}$$

We now use the above results and obtain the piecewise linear regions for each of the four SAEs defined previously.

E.2.1. ReLU, JumpReLU SAE

First note that the piecewise linear regions and receptive fields of ReLU and JumpReLU SAEs are the same—since in both cases, the gating appears through the heaviside step function ($ReLU(\boldsymbol{x}) = \boldsymbol{x} \odot \mathbb{I}(\boldsymbol{x} \geq 0)$). Thus, we develop the linear pieces and receptive fields only for ReLU, since the corresponding ones for JumpReLU are identical. The piecewise linear regions of latents in ReLU SAE are described by the following claim:

Claim E.2. *For a layer defined as in Eq. 29, $\mathcal{L}_\mathcal{M}$ is given as:*

$$\mathcal{L}_\mathcal{M} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{w}_m^T x + b_m \geq 0 \forall m \in \mathcal{M}, \boldsymbol{w}_q^T x + b_q < 0 \forall q \notin \mathcal{M}\} \tag{34}$$

Thus, $\mathcal{M}$ is an intersection of N half-spaces, and thus is a convex polytope which may be bounded or unbounded.

Proof. This is a consequence of the observation in Lemma E.7 and the definition of the relu model 29. □

Lemma E.10. *If $b = 0$ in Eq. 29, then $\mathcal{L}_\mathcal{M}$ are unbounded convex polytopes with only one corner at the origin and flat faces, i.e., they are (unbounded) hyperpyramids.*

Thus, bias plays an important role in ReLU layers, allowing piecewise linear regions that are convex polytopes with multiple corners anywhere in space. The greater flexibility in defining the pieces allows greater expressivity by capturing a larger class of functions. The following (somewhat obvious) claim describes the receptive fields of model 1 neurons.

Claim E.3. *In Model 1 (29), for a given neuron $k \in [n]$, the receptive field $\mathcal{F}_k$ is given as:*

$$\mathcal{F}_k = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{w}_k^T \boldsymbol{x} + b_k \geq 0\} \tag{35}$$

which is a half-space defined by the normal vector $\boldsymbol{w}_k$ and bias b_k.

This is a straightforward consequence of the definition of the ReLU model in Eq. 29.

E.2.2. TopK SAE

Claim E.4. *For a layer defined as in Eq. 31, $\mathcal{L}_{\mathcal{M}}$ is given as:*

$$\mathcal{L}_{\mathcal{M}} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{w}_m^T \boldsymbol{x} \geq \boldsymbol{w}_q^T \boldsymbol{x} \forall m \in \mathcal{M}, q \notin \mathcal{M}\} \tag{36}$$

Thus, $\mathcal{M}$ is an intersection of $K(N-K)$ half-spaces all passing through the origin, and thus is a convex polytope which may be bounded or unbounded. In fact, it is an unbounded hyperpyramid, with a corner at the origin and flat faces. The normals to these half-spaces are pairwise differences between active and inactive weight vectors.

This again follows from the Gating Lemma E.7.

Claim E.5. *In Model 2 (31), for a given neuron $k \in [n]$, the receptive field $\mathcal{F}_k$ is given as:*

$$\mathcal{F}_k = \cup_{\mathcal{M}:k \in \mathcal{M}} \mathcal{L}_{\mathcal{M}} \tag{37}$$

which is a union of hyperpyramids with a corner at the origin. Note that in typical implementations of TopK, a pre-encoder bias is included, so the corner of the hyperpyramids is at the pre-encoder bias.

E.2.3. SpaDE

Claim E.6. *For a layer defined as in Eq. 1, $\mathcal{L}_{\mathcal{M}}$ is given as:*

$$\mathcal{L}_{\mathcal{M}} = \left\{ \boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x} - \boldsymbol{w}_m\|_2^2 - \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \|\boldsymbol{x} - \boldsymbol{w}_j\|_2^2 - \frac{1}{\lambda |\mathcal{M}|} \begin{cases} \leq 0, & \text{if } m \in \mathcal{M} \\ > 0, & m \notin \mathcal{M} \end{cases} \right\} \tag{38}$$

$$= \left\{ \boldsymbol{x} \in \mathbb{R}^n : \right. \tag{39}$$

$$\left(\boldsymbol{w}_m^T - \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \boldsymbol{w}_j^T \right) \boldsymbol{x} - \left(\|\boldsymbol{w}_m\|_2^2 - \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \|\boldsymbol{w}_j\|_2^2 \right) + \frac{1}{\lambda |\mathcal{M}|} \left. \begin{cases} \geq 0, & \text{if } m \in \mathcal{M} \\ < 0, & m \notin \mathcal{M} \end{cases} \right\} \tag{40}$$

Thus, $\mathcal{M}$ is an intersection of N half-spaces, and thus a convex polytope. Note that the normal to each half space is now chosen in an input-adaptive fashion ($m \in \mathcal{M}$) and is locally centered using the mean of other nearby prototypes that are active, i.e., $\left(\boldsymbol{w}_m^T - \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \boldsymbol{w}_j^T \right)$ where $\mathcal{M}$ is input adaptive. An alternate interpretation is using the first equation above, which defines the region as the set of points whose distance to active prototypes is within a tolerance of the average distance to all active prototypes, while distance to inactive prototypes is larger than the average distance to active prototypes.

Proof. This is again a consequence of the definition of sparsemax (Martins & Astudillo, 2016). □

Claim E.7. *In SpaDE (32), for a given neuron $k \in [n]$, the receptive field $\mathcal{F}_k$ is given as:*

$$\mathcal{F}_k = \cup_{\mathcal{M}:k \in \mathcal{M}} \mathcal{L}_{\mathcal{M}} \tag{41}$$

which is a union of convex polytopes, each of which includes the latent k in the set of active indices $\mathcal{M}$. Due to the use of euclidean distances in choosing active indices, the receptive field is a union of convex polytopes in the vicinity of the prototype a_k of latent k. This incorporates the notion of locality and flexibility in receptive field shapes, allowing latents to capture nonlinearly separable concepts.

E.3. KDS and Sparse Coding

K-Deep Simplex (KDS) (Tasissa et al., 2023) is the sparse coding framework which forms the outer optimization in the SpaDE. While this is a different framework, in this section we show that it is general enough to capture the standard sparse coding, i.e., for data generated using standard sparse coding, there exists a corresponding KDS framework that could have generated the same data. Note that we may have to increase the latent dimension (number of dictionary atoms) by one to obtain the corresponding KDS framework. This is stated and proved (with a constructive proof) in the following theorem.

Theorem E.11 (KDS can capture standard sparse coding)**.** *Given data $\mathcal{D} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(P)}\}$ generated from a standard sparse coding generative model, i.e., $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{z} + \eta$, where dictionary atoms (columns of $\boldsymbol{D}$) have unit norm and $\boldsymbol{z}$ is unconstrained, there exists a scaling of the data such that it can be represented using the K-Deep Simplex (Tasissa et al., 2023) framework, i.e., $\tilde{\boldsymbol{x}} = \kappa\boldsymbol{x} = \tilde{\boldsymbol{D}}\tilde{\boldsymbol{z}} + \tilde{\eta}$, where $\tilde{\boldsymbol{z}} \in \Delta^s$.*

Proof. Consider the following scalar:

$$\kappa = \left(\max_{\boldsymbol{x}\in\mathcal{D}} \sum_i z_i(\boldsymbol{x})\right)^{-1}$$

Normalizing data using κ above gives us,

$$\begin{aligned}\tilde{\boldsymbol{x}} &= \kappa\boldsymbol{x}\\ &= \boldsymbol{D}\frac{\boldsymbol{z}}{\max_{\boldsymbol{x}\in\mathcal{D}}\sum_i z_i(\boldsymbol{x})} + \kappa\eta\\ &= \boldsymbol{D}\hat{\boldsymbol{z}} + \tilde{\eta}\end{aligned}$$

By definition, $\hat{\boldsymbol{z}}$ defined above always satisfies $\sum_i \hat{z}_i \leq 1$, so let $\beta = 1 - \sum_i \hat{z}_i$. Appending an all-zeros dictionary atom to $\boldsymbol{D}$, $\tilde{\boldsymbol{D}} = [\boldsymbol{D}, \boldsymbol{0}]$ and assigning the residual to $\tilde{\boldsymbol{z}} = [\hat{\boldsymbol{z}}^T, \beta]^T$ gives us the following:

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{D}}\tilde{\boldsymbol{z}} + \tilde{\eta}, \quad \text{where } \tilde{\boldsymbol{z}} \in \Delta^s$$

implying that the original data can be represented in the framework of KDS. □

E.4. SpaDE

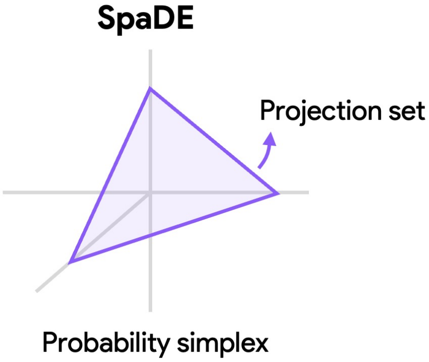

Figure E.2: **SpaDE shows adaptive sparsity by projecting onto the probability simplex**. In this illustrative $3D$ figure, note $\|\boldsymbol{x}\|_0 = 3$ for points on the face, $\|\boldsymbol{x}\|_0 = 2$ for points on edges along subspaces, and $\|\boldsymbol{x}\|_0 = 1$ for corners on coordinate axes.

Sparsemax is a projection onto the probability simplex, which can be written as (see Proposition 1 in (Martins & Astudillo, 2016))

$$\text{Let } \boldsymbol{z} = \text{Sparsemax}(\boldsymbol{y}) \tag{42}$$

$$\text{Then, } z_i = \text{ReLU}(y_i - \frac{1}{|\mathcal{M}|}\sum_{j\in\mathcal{M}} y_j + \frac{1}{|\mathcal{M}|}) \tag{43}$$

SpaDE is defined using squared euclidean distances between an input vector and some *prototypes* (or landmarks) in input space (Eq. 1), which gives us

$$y_i = -\lambda|\boldsymbol{x} - \boldsymbol{w}_i|_2^2 \tag{44}$$

$$\implies \text{Sparsemax}(\boldsymbol{y})_i = \text{ReLU}\left(2\lambda(\boldsymbol{w}_i - \frac{1}{|\mathcal{M}|}\sum_j \boldsymbol{w}_j)^T\boldsymbol{x} - \lambda(|\boldsymbol{w}_i|^2 - \frac{1}{|\mathcal{M}|}|\boldsymbol{w}_j|^2) + \frac{1}{|\mathcal{M}|}\right) \tag{45}$$

$$= ReLU(\tilde{\boldsymbol{W}}(\boldsymbol{x})\boldsymbol{x} + \tilde{\boldsymbol{b}}_e(\boldsymbol{x})) \tag{46}$$

where $\tilde{\boldsymbol{W}}(\boldsymbol{x}) = 2\lambda(\boldsymbol{w}_i - \frac{1}{|\mathcal{M}|}\sum_j \boldsymbol{w}_j)$, $\tilde{\boldsymbol{b}}_e = -\lambda(|\boldsymbol{w}_i|^2 - \frac{1}{|\mathcal{M}|}|\boldsymbol{w}_j|^2) + \frac{1}{|\mathcal{M}|}$ and $\mathcal{M}$ is the set of active indices, which is uniquely determined by the constraint $\sum_i \text{Sparsemax}(\boldsymbol{y})_i = 1$ (see Proposition 1 in (Martins & Astudillo, 2016) for uniqueness). Note that $\tilde{\boldsymbol{W}}(\boldsymbol{x}), \tilde{\boldsymbol{b}}_e(\boldsymbol{x})$ are both *piecewise constant* on regions of input space marked by the same choice of active indices.

Thus, SpaDE is equivalent to a ReLU SAE, but with a linear transformation and bias that are input-adaptive (piecewise constant). SpaDE is thus piecewise linear and continuous (continuity follows from the continuity of sparsemax). Note that this is a nontrivial result: despite appearing quadratic in input due to the use of squared euclidean distances, SpaDE is a piecewise linear function of the input. This result is also exact, and is NOT a first order Taylor series approximation.

However, SpaDE differs from a ReLU SAE by using linear transformations defined with respect to a local origin, which is uniquely determined by the set of active SAE latents, similar to recent work on steering (Wu et al., 2025b).

Since SAEs are completely described by their inner and outer optimization problems (see Theorem E.1), we now describe these components for SpaDE.

The inner optimization (Eq. 10) for the SpaDE is as follows:

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{\pi}, \boldsymbol{W}, \boldsymbol{x}) &= \sum_i \pi_i \|\boldsymbol{x} - \boldsymbol{w}_i\|_2^2 + \frac{1}{2\lambda}\|\boldsymbol{\pi}\|_2^2 \\ \mathcal{S} &= \{\boldsymbol{\pi} \in \mathbb{R}^s : \ \pi_i \geq 0, \sum_i \pi_i = 1\} \end{aligned} \tag{47}$$

This resembles one-sided optimal transport with a squared 2-norm regularizer. This problem is one-sided because there is no constraint on how much *weight* sits on each prototype across different inputs (optimization is performed independently for each input). The squared $2-$norm regularizer is known to lead to sparse transport plans in the optimal transport literature (see (Liu et al., 2022)).

The outer optimization for SpaDE (Eq. 10) is a locality-enforced version of dictionary learning called K-Deep Simplex (KDS) (Tasissa et al., 2023). In this framework, the sparse code is constrained to belong to the probability simplex, i.e., $\boldsymbol{z} \in \Delta^s = \{\boldsymbol{y} \in \mathbb{R}^s : \sum_i y_i = 1, y_i \geq 0 \forall i\}$, while the dictionary atoms $\boldsymbol{D}$ are unconstrained. The distance-weighted L1 regularizer encourages each datapoint to use those dictionary atoms which are close to itself in euclidean distance, inducing a soft clustering bias. Even though this is a different dictionary learning framework than standard sparse coding, it is expressive enough to capture the standard sparse coding setup, i.e., for any standard sparse coding problem, there exists an equivalent KDS problem (see Theorem E.11 in Appendix).

While this outer optimization (KDS) is a different problem than the standard dictionary learning problem, it may be useful for interpretability since it has the following advantages:

1. It avoids shrinkage, since the L1 norm of the sparse representation $\boldsymbol{z}(\boldsymbol{x})$ is constrained to equal 1 for all inputs
2. Constraining the sparse code to the probability simplex finds support in an oft-cited paper demonstrating the linear representation hypothesis in word embeddings under a random-walk based generative model of language (Arora et al., 2018). Their main result (Theorem 2) shows that representations are *convex* combinations of concepts, as opposed to unconstrained linear combinations, which is better interpreted as assigning vectors (with magnitude and direction; alternatively, locations) to concepts rather than directions (without magnitude). This idea of concepts as vectors has also been demonstrated both theoretically and empirically in the final layer representations of language models (Park et al., 2024).

Note how SpaDE satisfies the two data properties of nonlinear separability and heterogeneity:

1. The projection set $\mathcal{S}$ in SpaDE is the probability simplex, which admits edges/corners with varying levels of sparsity, thereby allowing the representation of heterogeneous concepts. For any choice of $k \in \{1, 2, ..., s\}$, there are $\binom{s}{k}$ choices of indices $\mathcal{M}_k$ for a k-sparse representation, and points $\{\boldsymbol{x} \in \mathbb{R}^s : x_i = 0, i \notin \mathcal{M}_k, \sum_{j \in \mathcal{M}_k} x_j = 1, x_j \geq 0\} \subseteq \Delta^s$ which admit this level of sparsity, thereby capturing concept heterogeneity.
2. The receptive fields of SpaDE (see App. E.2.3) are local to each prototype (encoder weight vector), and are flexibly defined as the union of convex polytopes. This allows latents in SpaDE to become monosemantic to concepts which are nonlinearly separable from the rest of the data.

F. Further Results

In this section, we present a more detailed analysis of the results from each of our four experiments.

F.1. Separability Experiment

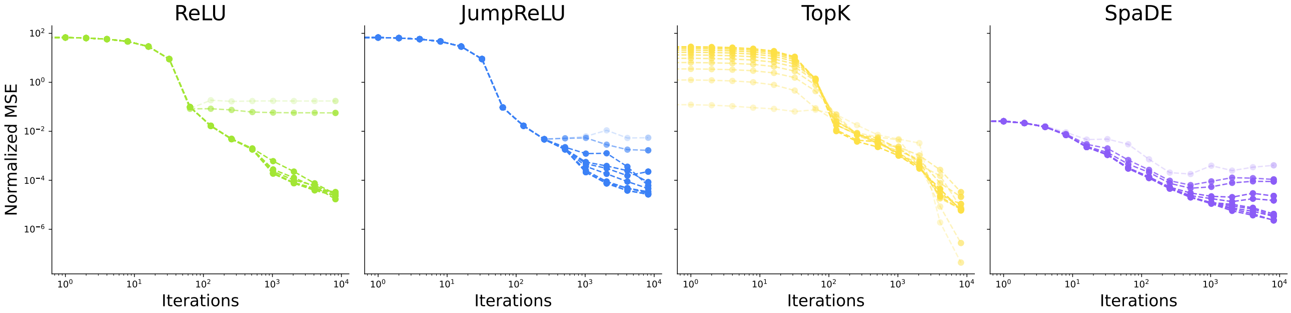

Figure F.3: Evolution of normalized MSE with training iterations for various SAEs on the *separability experiment*. Color intensity is proportional to L_0 (darker colors imply more dense SAE latents).

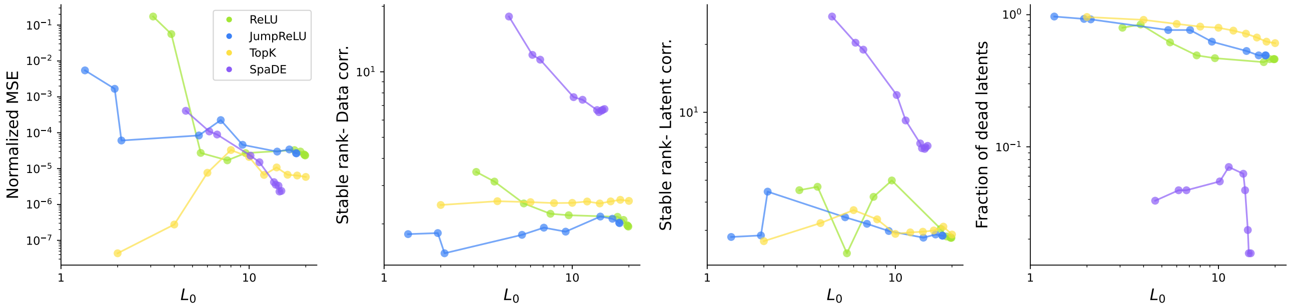

Figure F.4: Normalized MSE (normalized with variance of data), Stable ranks (of data correlations, latent correlations matrices), and fraction of dead latents as a function of sparsity (L_0) for the *separability experiment* (Sec. 4.1)

Fig. F.3 shows the evolution of normalized MSE (NMSE- MSE normalized by the variance of data) with training iterations for each SAE, for different levels of sparsity. Note that denser representations (higher L_0 and thus darker colors in Fig. F.3) lead to lower NMSE. While all SAEs end up at similar levels of NMSE, their ability to extract concepts from data is markedly different (as described in Sec. 3). A per-concept breakdown of training dynamics is shown in Fig. F.5. For comparison, this figure also includes the mean of the squared norm of each concept (which equals MSE if the SAE predicts the origin for all inputs), variance of each concept (which equals MSE if the SAE predicts the mean of each concept). Thus, SAEs whose MSE saturates at the concept variance are likely to be predicting the mean of the concept for all points, whereas when MSE goes below concept variance, the SAE explains within-concept variance. Also shown in gray is MSE with respect to the center of each concept, which ideally must match concept variance if the SAE reconstructs all points (which is observed in most cases).

In Fig. F.4, final NMSE as a function of sparsity (L_0) shows that while all SAEs have comparable MSE-sparsity curves at dense representations (high L_0), TopK's NMSE goes down significantly more than others. This is a consequence of TopK learning a redundant solution, by just using two latents as an orthogonal basis to represent all data. Fraction of dead latents show large numbers of dead latents at high sparsity levels for ReLU, JumpReLU and TopK, with this going down (exponentially) as representations become more dense. However, SpaDE shows significantly fewer dead latents at all levels of sparsity. Stable ranks of cosine similarities between latent representations of pairs of data points (data corr.), and between pairs of latents across all data points (latent corr.) show that SpaDE has very high stable ranks, indicating high specialization of latents. The other SAEs have comparable stable ranks, all much lower than the desirable stable rank of 6 (equal to the number of clusters in data).

The SAE latent activation profiles for each concept are shown as histograms in Fig. F.6. While variations exist across concepts, there is a common structure to the profiles for each SAE (SpaDE appears *pointy*, indicating a second mode other than zero).

Cosine similarities between latent representations of pairs of data points are shown for different levels of sparsity in Fig.

F.7. Notice that SpaDE has the lowest cross-concept correlations of all SAEs, and these correlations do not decrease much especially in ReLU and JumpReLU. The corresponding figure with similarities between pairs of latents across all datapoints is in Fig. F.8. Here, the number of dead latents increases with increasing sparsity, leading to very few active latents (only active latents are shown). Broadly, note the decrease in co-occurrences with increase in sparsity- also note how ReLU and JumpReLU result in newer correlation structures with greater sparsity.

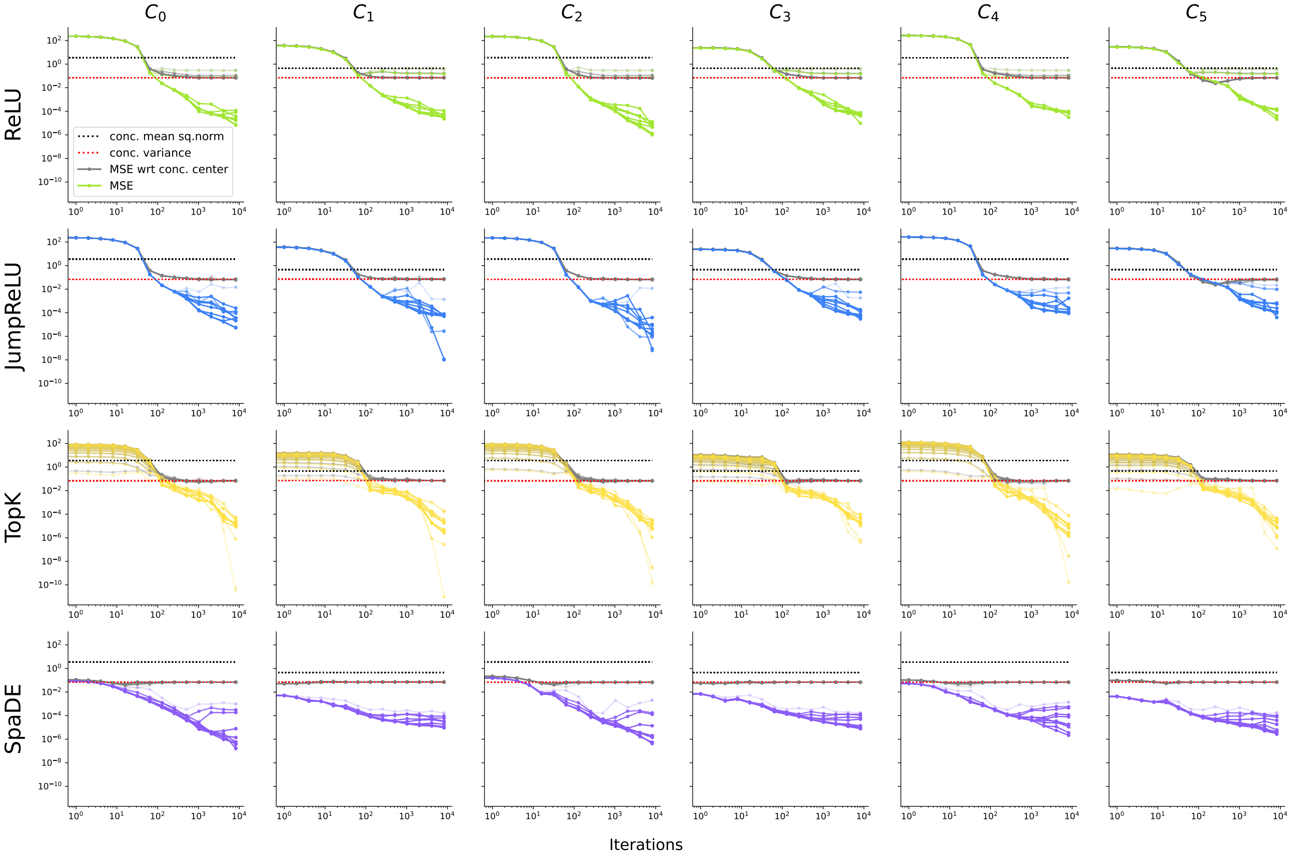

Figure F.5: Training dynamics for each concept (column) across SAEs (rows) for *separability experiment*: colored solid lines are MSE, with intensity of color proportional to L_0. Gray lines show MSE of SAE predictions with respect to the center of each cluster; intensity is again proportional to L_0. . Black dotted line shows the mean squared norm of each cluster, which would equal the MSE if the SAE predicted the origin for all datapoints. Red dotted line shows variance of each cluster, which again equals MSE if an SAE predicts the center of the cluster. Note that when a model reconstructs data well, MSE wrt cluster center equals the variance of the cluster (as observed here)

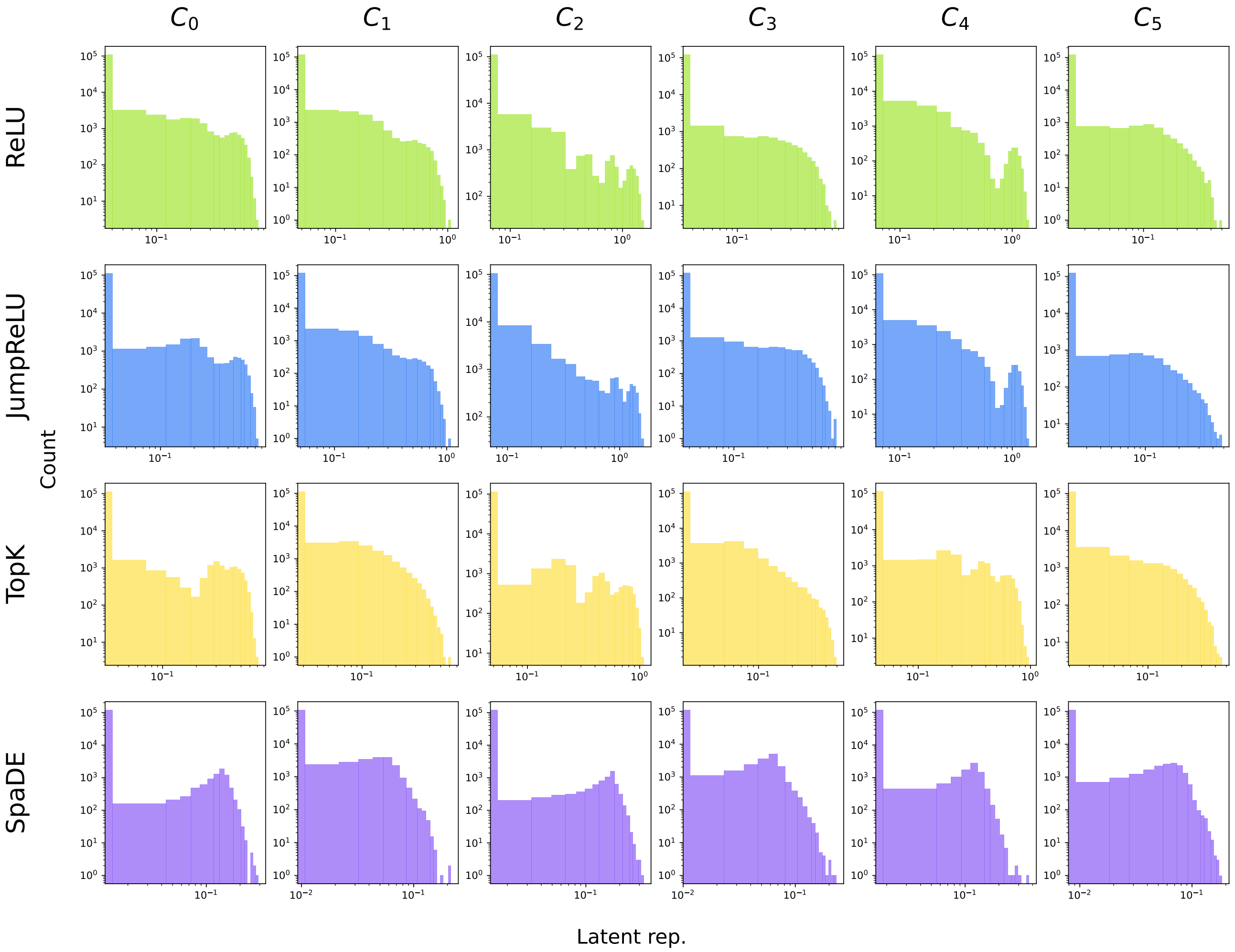

Figure F.6: Histogram of latent representations for each concept of various SAEs on the *separability experiment*.

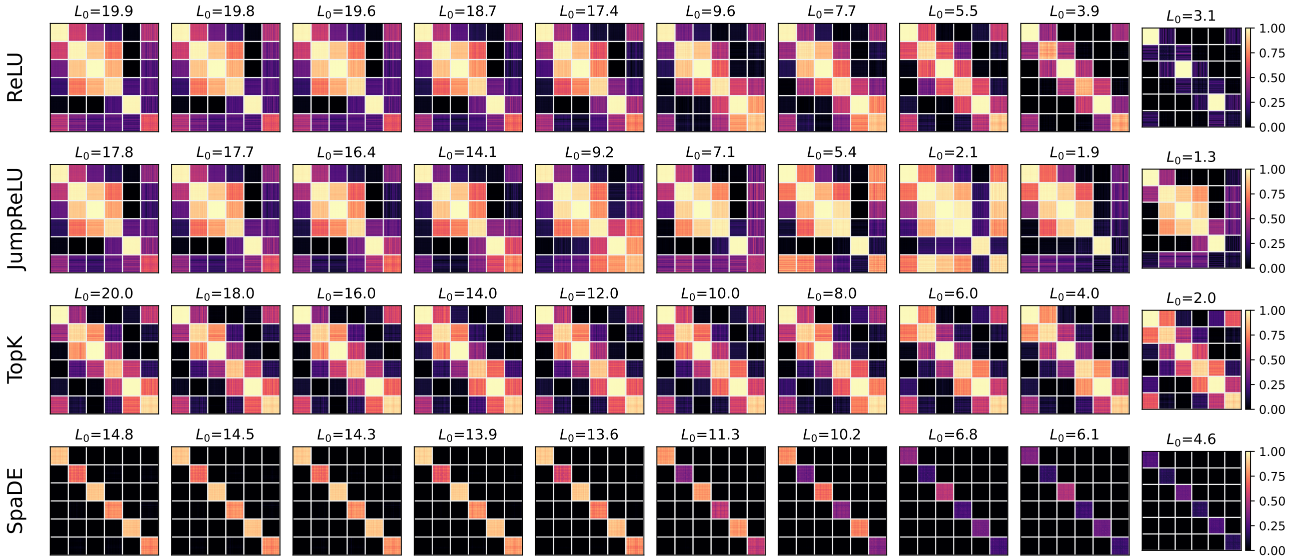

Figure F.7: Data correlations for various sparsity levels on the *separability experiment*: Pairwise cosine similarities between SAE latent representations of datapoints. White lines separate different concepts.

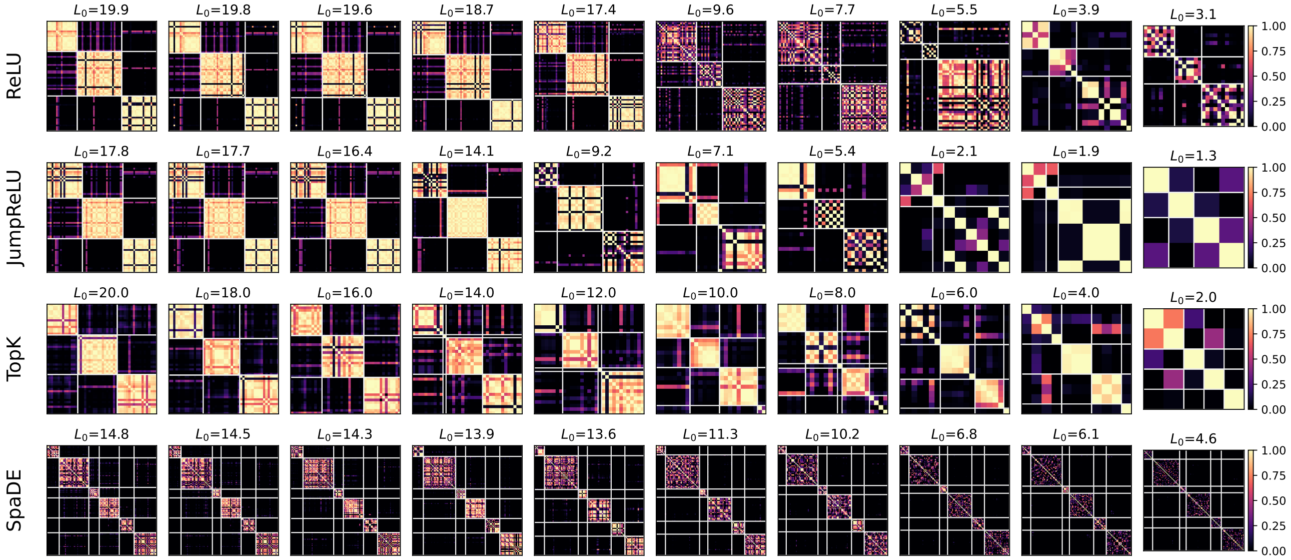

Figure F.8: Latent correlations for various sparsity levels on the *separability experiment*: Pairwise cosine similarities: pairwise cosine similarities between different SAE latents, computed across data from all concepts.

F.2. Heterogeneity Experiment

The overall training dynamics (on data from all concepts) is shown in Fig. F.10- note, again, that for low sparsity (high L_0, darker color) all SAEs reach similar levels of NMSE, but differ for higher sparsity levels. The per-concept breakdown of MSE, and comparison with mean squared norm, concept variance and MSE with respect to the center of each concept is in Fig. F.5 . The *kink* in gray lines is precisely the point where the model transitions from learning to represent the mean, to learning to explain the within-concept variance, clearly demonstrating two phases in learning: learning the right *scale* for the data (since initial model predictions may not match the true scale of data), thereby predicting the mean well, followed by learning the distribution of the data.

Fig. F.12 shows latent activation profiles for each concept and each SAE ($k = 32$ in TopK). Since TopK with $k = 32$ cannot allocate enough latents for large intrinsic dimension concepts, it increases activations on smaller number of concepts instead. Cosine similarities between SAE latent representations for pairs of data points, and pairs of latents across all datapoints, is shown for varying levels of sparsity in Fig. F.13, F.14 respectively. All SAEs (except JumpReLU) do a decent job at reducing correlations between pairs of data points, but in the latent correlation plots, we see how TopK fails to adaptively allocate latents to heterogenous concepts, especially at moderate levels of sparsity, while the other SAEs do well- have different sized blocks in block-structured matrix.

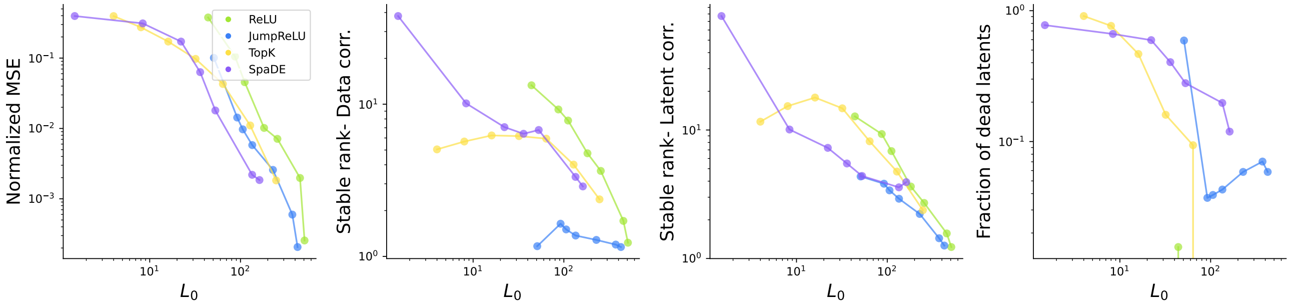

Figure F.9: Normalized MSE (normalized with variance of data), Stable ranks (of data correlations, latent correlations matrices), and fraction of dead latents as a function of sparsity (L_0) for the *heterogeneity experiment* (Sec. 4.2).

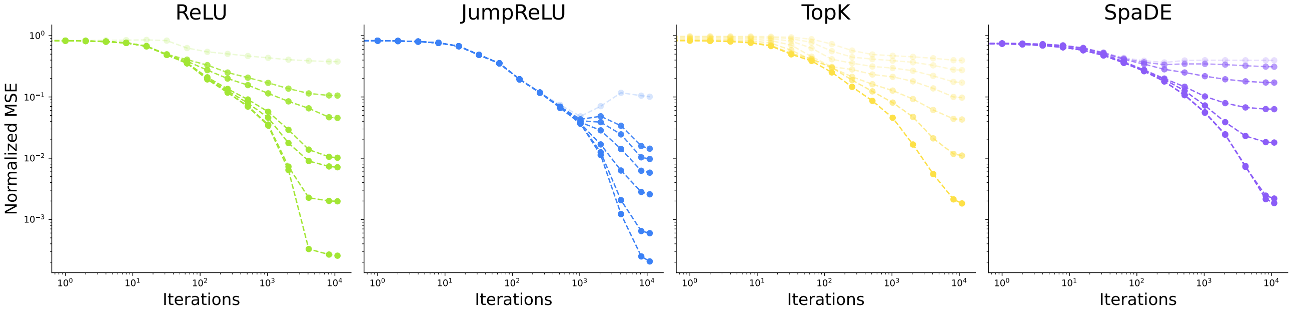

Figure F.10: Evolution of normalized MSE with training iterations for various SAEs on the *heterogeneity experiment*. Color intensity is proportional to L_0 (darker colors imply more dense SAE latents).

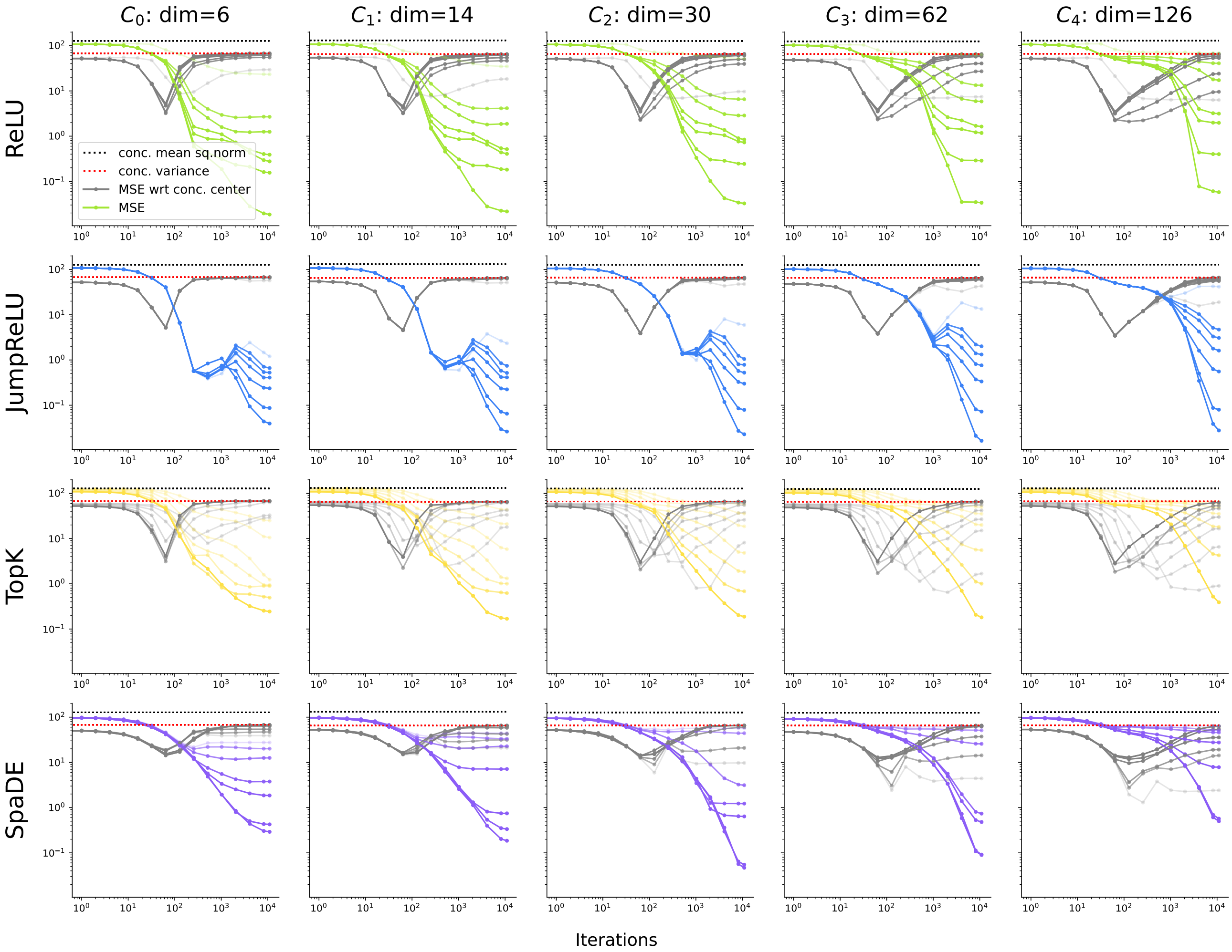

Figure F.11: Training dynamics for each concept (column) across SAEs (rows) for *heterogeneity experiment*: colored solid lines are MSE, with intensity of color proportional to L_0. Gray lines show MSE of SAE predictions with respect to the center of each cluster; intensity is again proportional to L_0. . Black dotted line shows the mean squared norm of each cluster, which would equal the MSE if the SAE predicted the origin for all datapoints. Red dotted line shows variance of each cluster, which again equals MSE if an SAE predicts the center of the cluster. Note that when a model reconstructs data well, MSE wrt cluster center equals the variance of the cluster (as observed here)

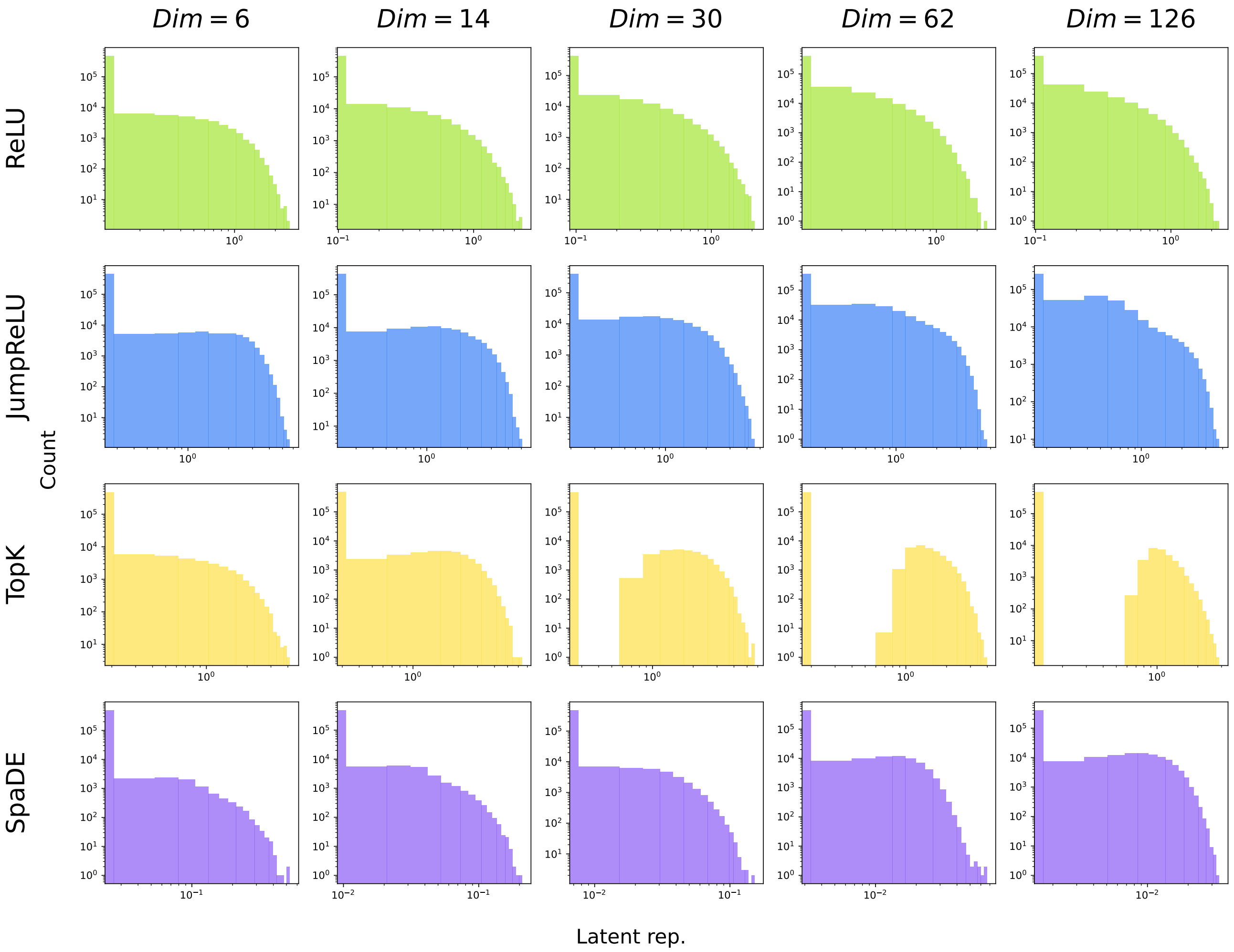

Figure F.12: Histogram of latent representations for each concept of various SAEs on the *heterogeneity experiment*.

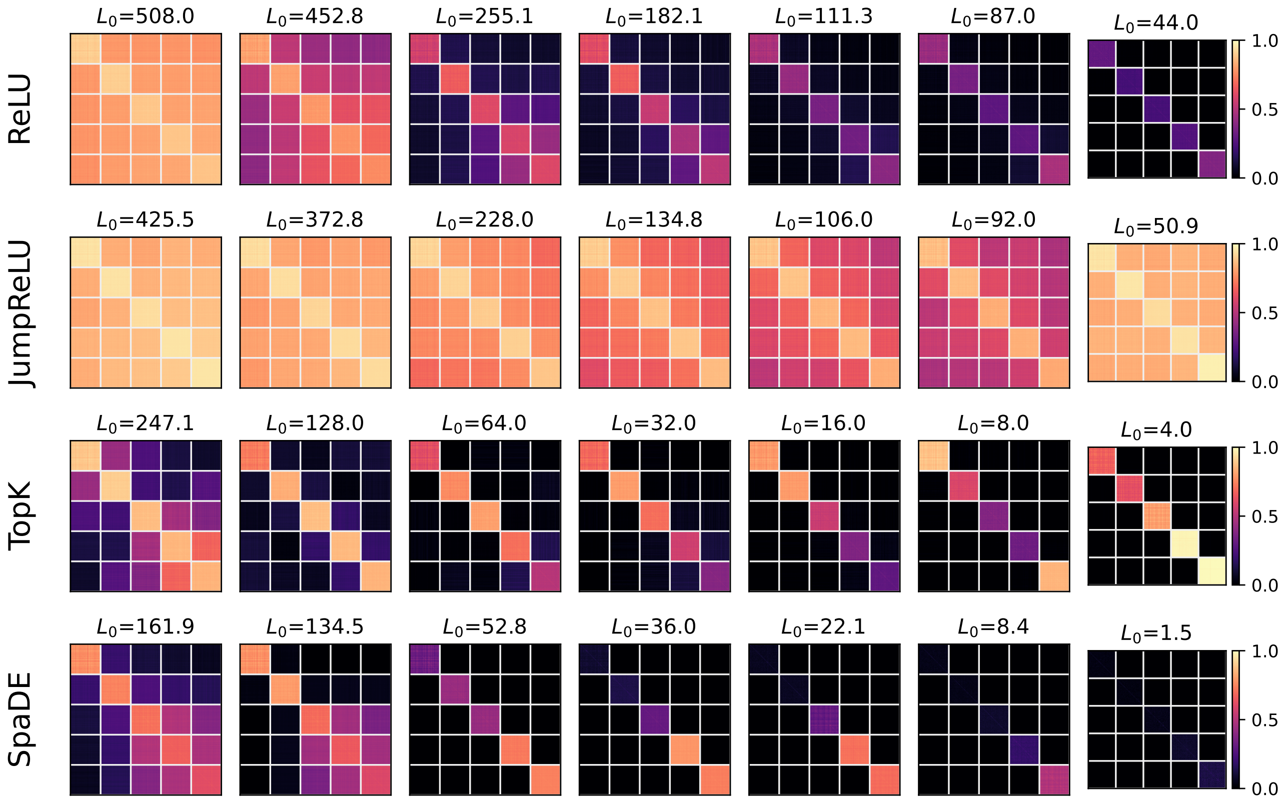

Figure F.13: Data correlations for various sparsity levels on the *heterogeneity experiment*: Pairwise cosine similarities between SAE latent representations of datapoints. White lines separate different concepts.

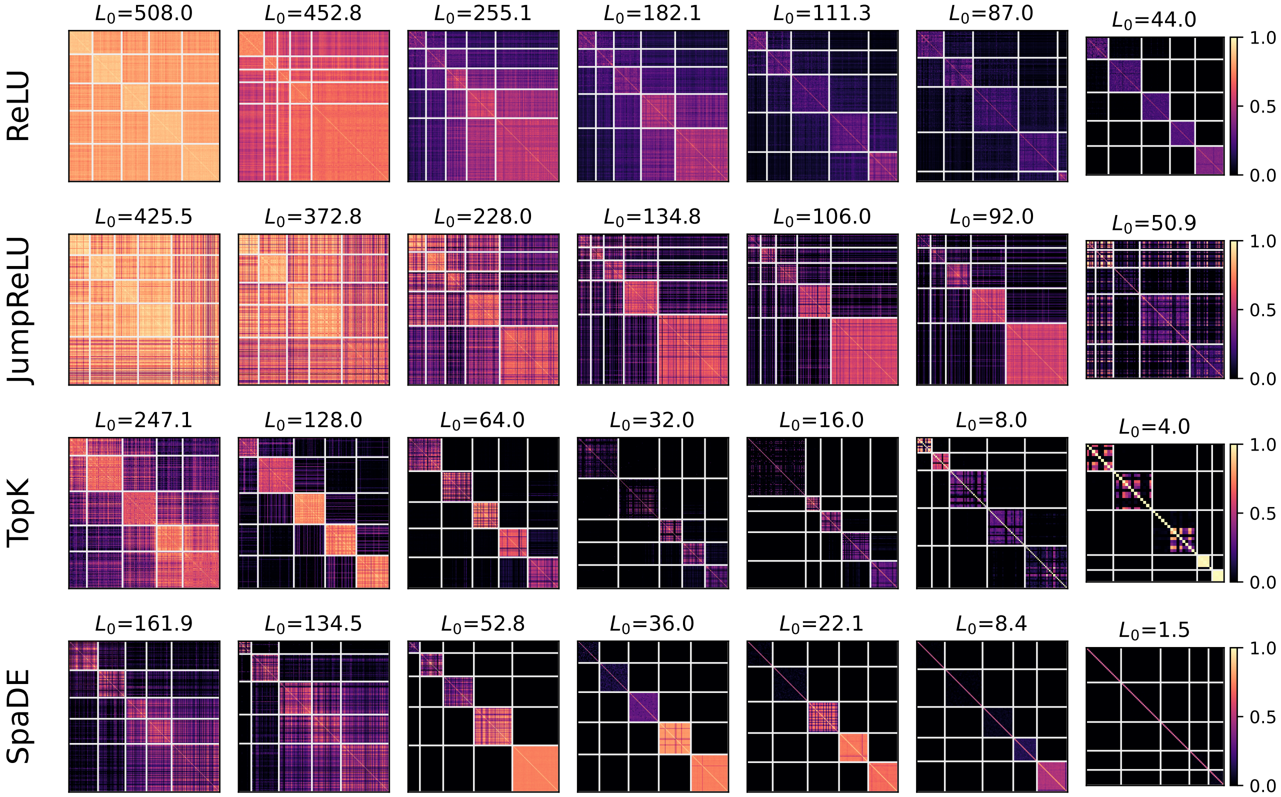

Figure F.14: Latent correlations for various sparsity levels on the *heterogeneity experiment*: Pairwise cosine similarities: pairwise cosine similarities between different SAE latents, computed across data from all concepts.

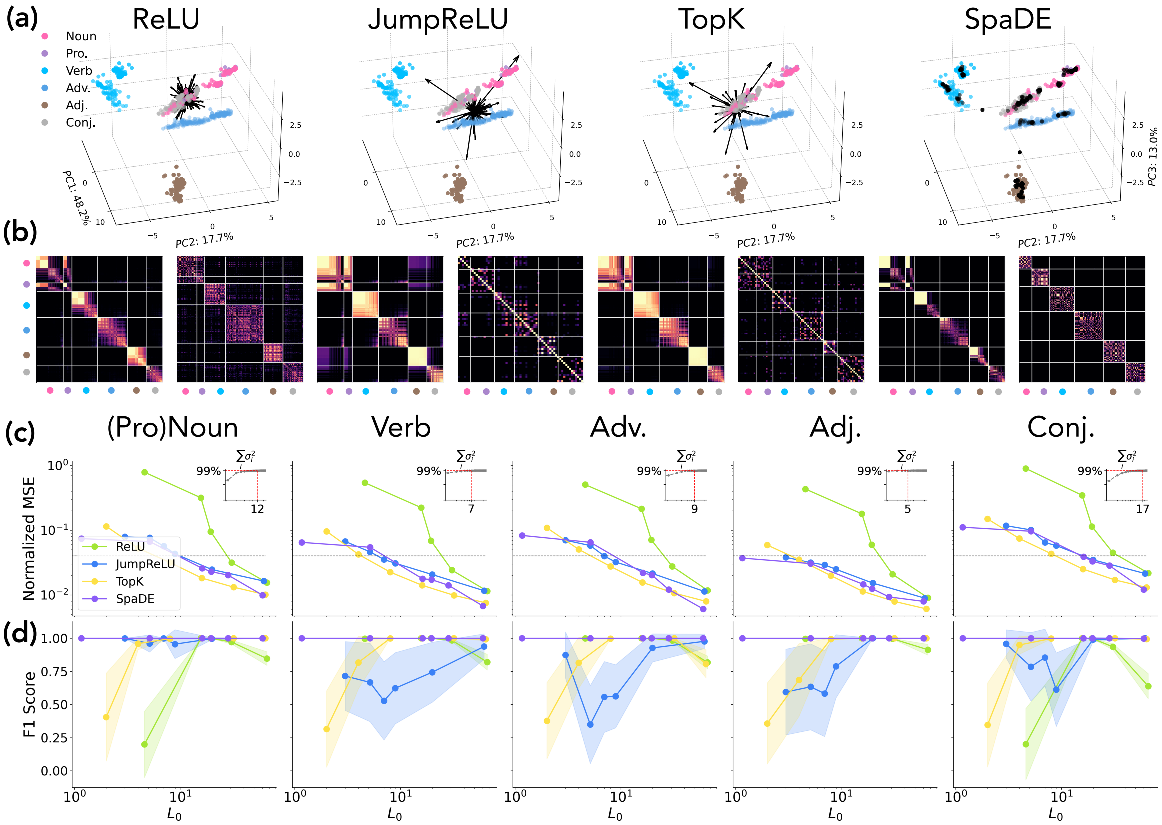

Figure F.15: **Investigating SAE properties on GPT for formal languages**. **(a)** 3D PCA of model activations and SAE encoder weights, where datapoints are colored by part-of-speech (PoS). Encoder weights are indicated by points for SpaDE and arrows for the other SAEs. **(b)** Matrix of cosine similarities between pairs of data and pairs of latents (in order) for each SAE. White lines separate different PoS. **(c)** MSE normalized by PoS variance as a function of sparsity, for each PoS. *Inset:* cumulative sum of variance (eigenvalues of data correlations) of each PoS, where the effective dimension (variance $> 99\%$) of each PoS is shown. **(d)** Top-20 F_1-scores for different PoS from each SAE's latents (a measure of monosemanticity).

F.3. Formal Language Experiments

Dataset and Experiment: Building on recent work using formal languages for making predictive claims about language models (Jain et al., 2023; Lubana et al., 2024; Allen-Zhu & Li, 2023), we use this setting as a semi-synthetic setup for corroborating our claims. Specifically, we analyze the English PCFG with subject-verb-object sentence order proposed in (Menon et al., 2024). We train 2-layer Transformers (Andrej Karpathy, 2023) from scratch on strings of maximum length 128 tokens from the formal grammar above. SAEs are then trained on activations retrieved from the middle residual stream of the model.

Observations: Results are shown in Fig. F.15. Different parts of speech (PoS), the core concepts of the grammar, form clusters in a 3D PCA of their representations (see row (a)). SpaDE learns to tile the PoS clusters well. While all SAEs do a good job at making their latents uncorrelated across PoS (first column per SAE, row (b)), there are co-occurring latents across PoS in all SAEs except SpaDE (second column per SAE, row (b)). PoS seem to have different intrinsic dimensions (number of dimensions to capture 99% of total variance in data, inset in row (c)), which leads to TopK requiring different values of K to explain the data (crosses 5% normalized mse with differing values of k, row (c)). PoS also appear to have differing levels of linear separability, as ReLU and JumpReLU show lower F1 scores which peak at different levels of sparsity for each concept (row (d)), while SpaDE shows a perfect F1 score of 1 in its most monosemantic latents.

Furthermore, we report several more results in the formal language experimental setup. Specifically, we show how with changing sparsity of the latent code, fidelity metrics, e.g., normalized MSE scales changes and stable rank of both data and latent correlations changes (Fig. F.16); how monosemanticity changes, i.e., how F1 scores averaged across latents and the concept they achieve maximum F1 score on change (indicating their specialization to that concept) (Fig. F.17 Left); and how percentage of dead latents change (Fig. F.17 Right). These results are repeated at the concept-level, i.e., at the level of parts-of-speech, in Figs. F.18, F.19. Inline with results on heatmaps demonstrating correlation between sparse codes of samples from different concepts and between vector denoting which samples a given latent activates for, we retrieve results in Fig. F.20, F.21. The results above are perfectly inline with our findings from the main paper, e.g., that SpaDE achieves highly monosemantic features. The new and intriguing results involve demonstrations of how effective SpaDE can be at discerning position of a concept (part-of-speech) in a sentence, when compared to other protocols which learn a more uniform representation.

Further, we also provide 2D and 3D PCA visualizations of different SAEs' retrieved latents in two different manners: (i) assess which datapoints a latent activates for and project it into a low-dimensional space identified using PCA, and (ii) assess which latents a datapoint activates, and project this activation vector. The former helps assess how monosemantic latents are, i.e., whether they activate for specific concepts, and the latter helps assess how specific latents are, i.e., whether a datapoint only activates a specific latent and hence there is no regularity present. Results show most SAEs, when they perform well, organize latents in a very structured manner (like a tetrahedron), but SpaDE succeeds at this throughout.

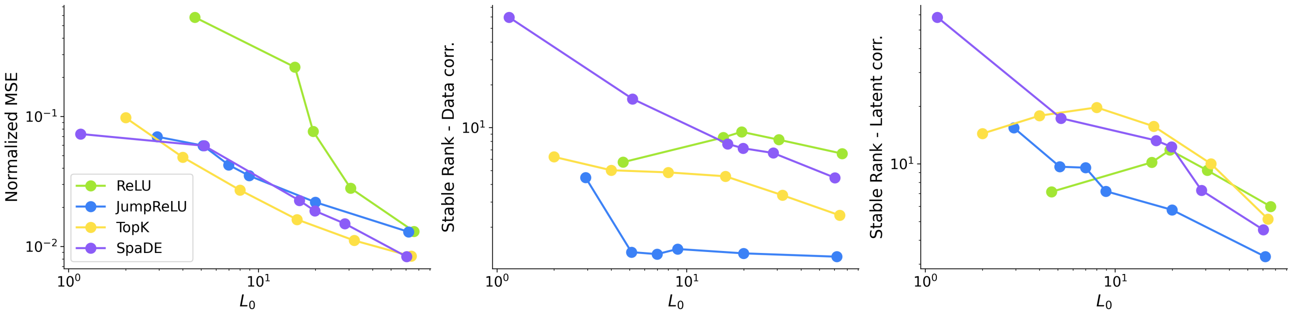

Figure F.16: Normalized MSE and Stable ranks as a function of sparsity in the Formal Language setup.

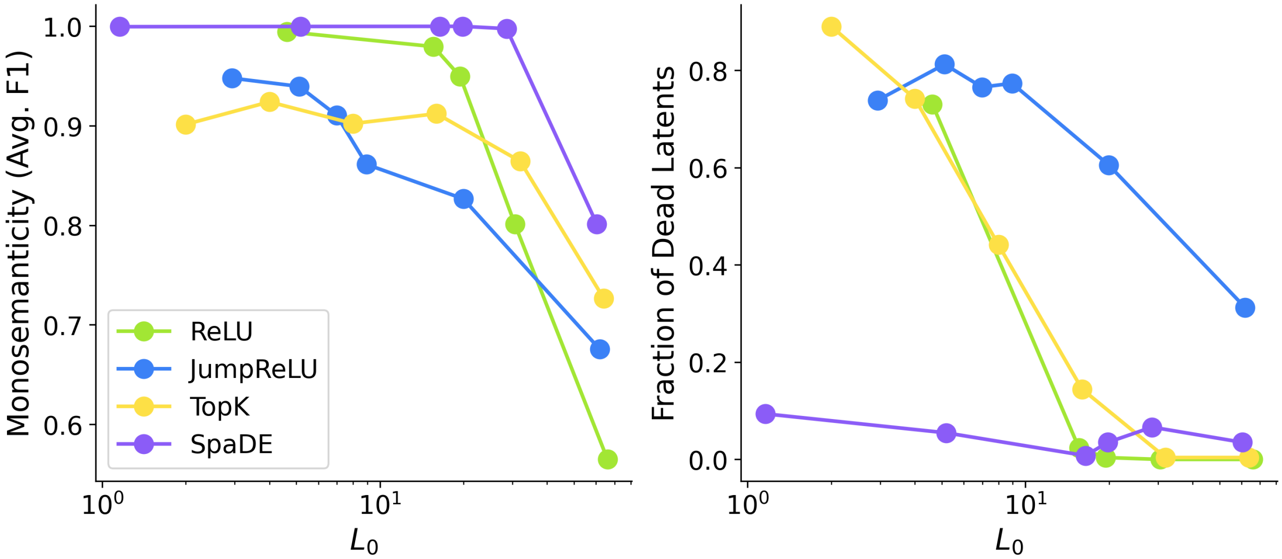

Figure F.17: Monosemanticity (F1 scores averaged over latents) and fraction of dead latents as a function of sparsity for different SAEs in the Formal Language setup.

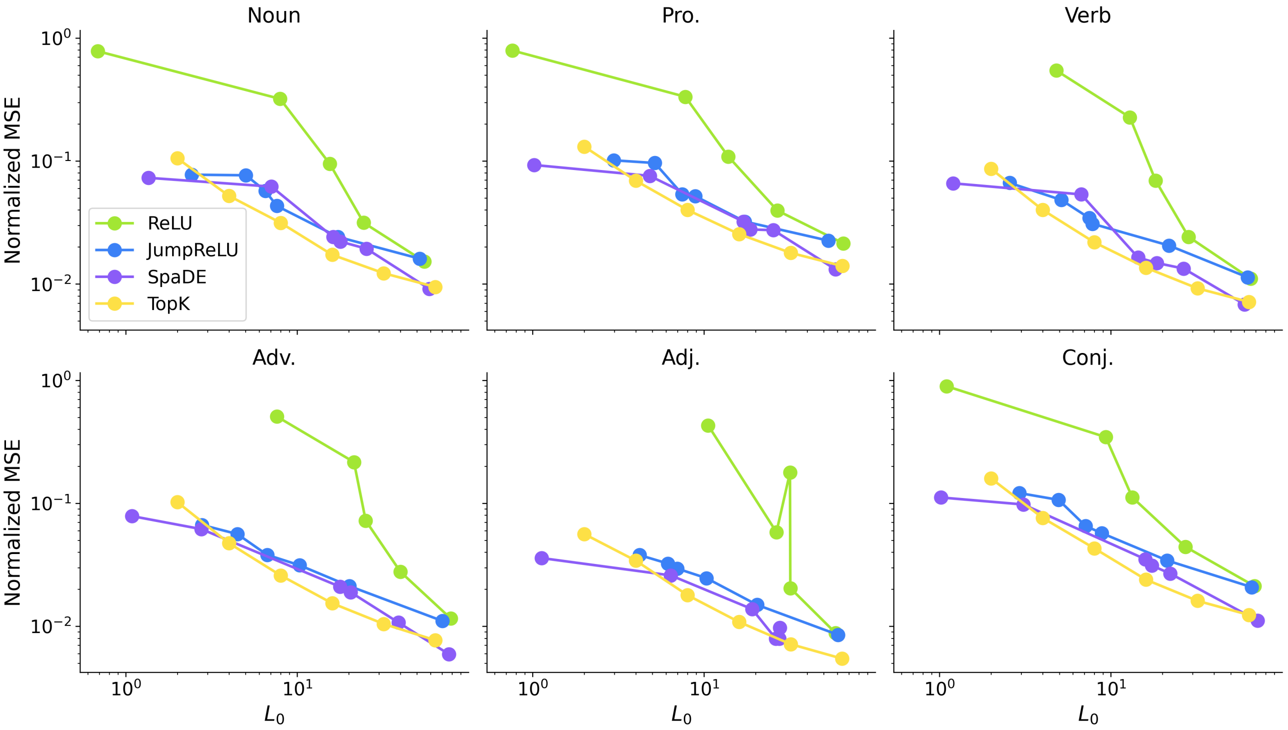

Figure F.18: Normalized MSE decomposed by concepts (parts-of-speech) and plotted as a function of sparsity in the Formal Language setup.

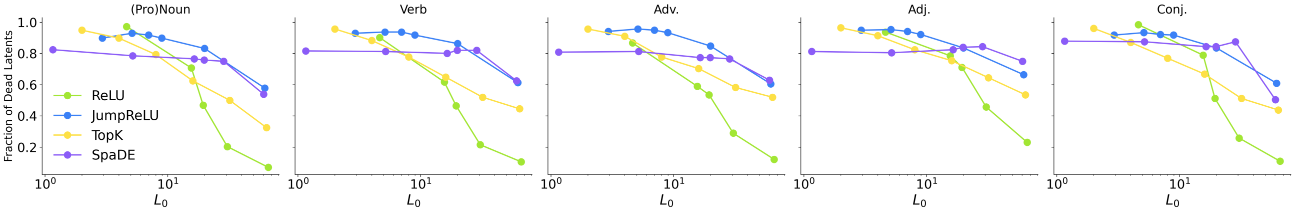

Figure F.19: Percentage of Dead Latents decomposed by concepts (parts-of-speech) and plotted as a function of sparsity in the Formal Language setup. Note that in such a concept-conditioned count of dead latents, one ends up counting both the latents that are always inactive and ones that are inactive for the specific concept under consideration.

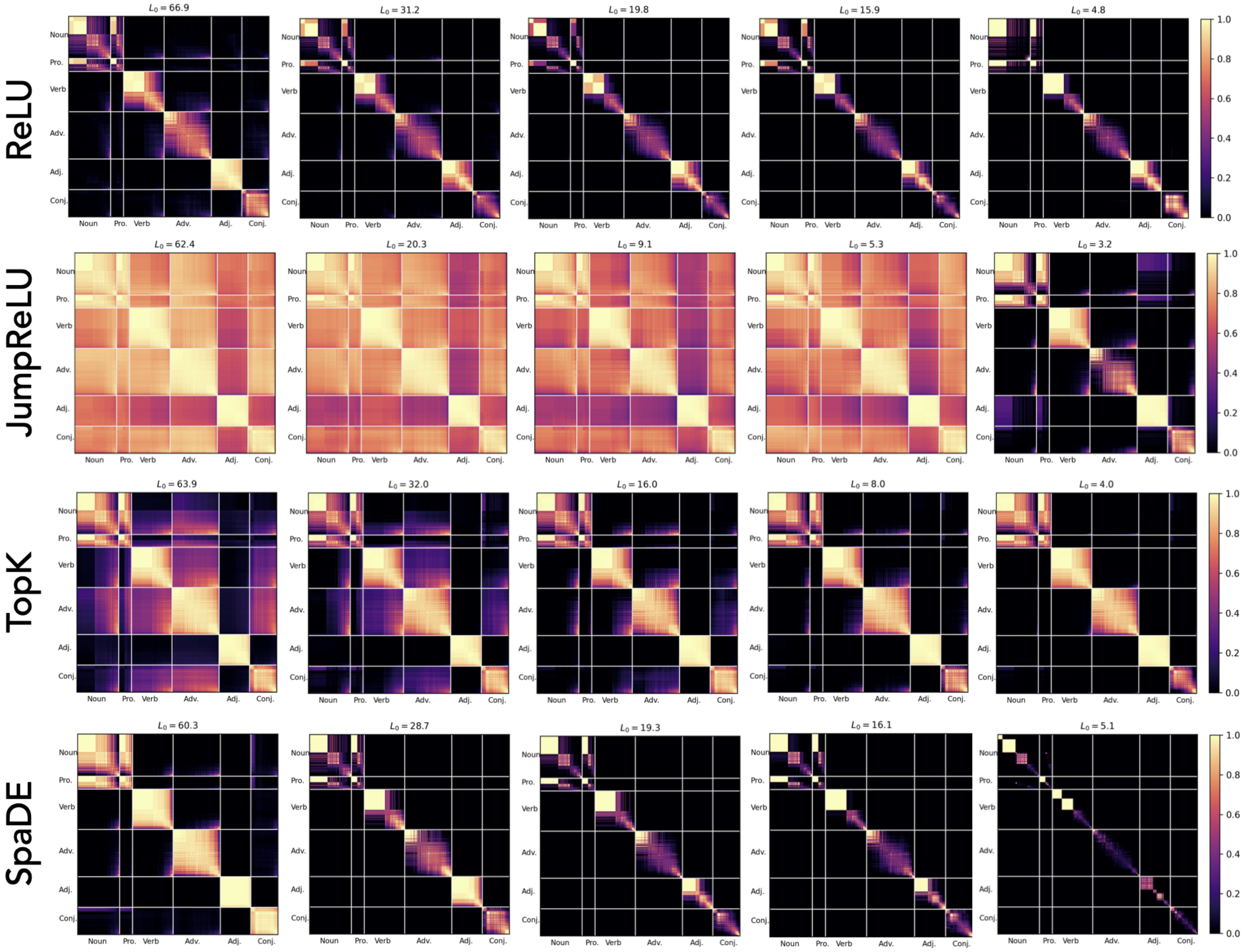

Figure F.20: Correlation between sparse codes of different concepts (parts-of-speech) in the Formal Language setup. Datapoints for different concepts are sorted according to which concept they come from (using a predefined order on the parts-of-speech) and according to their position in a sentence, hence highlighting position dependence. Lines demarcate boundaries at which tokens corresponding to different concepts start / end.

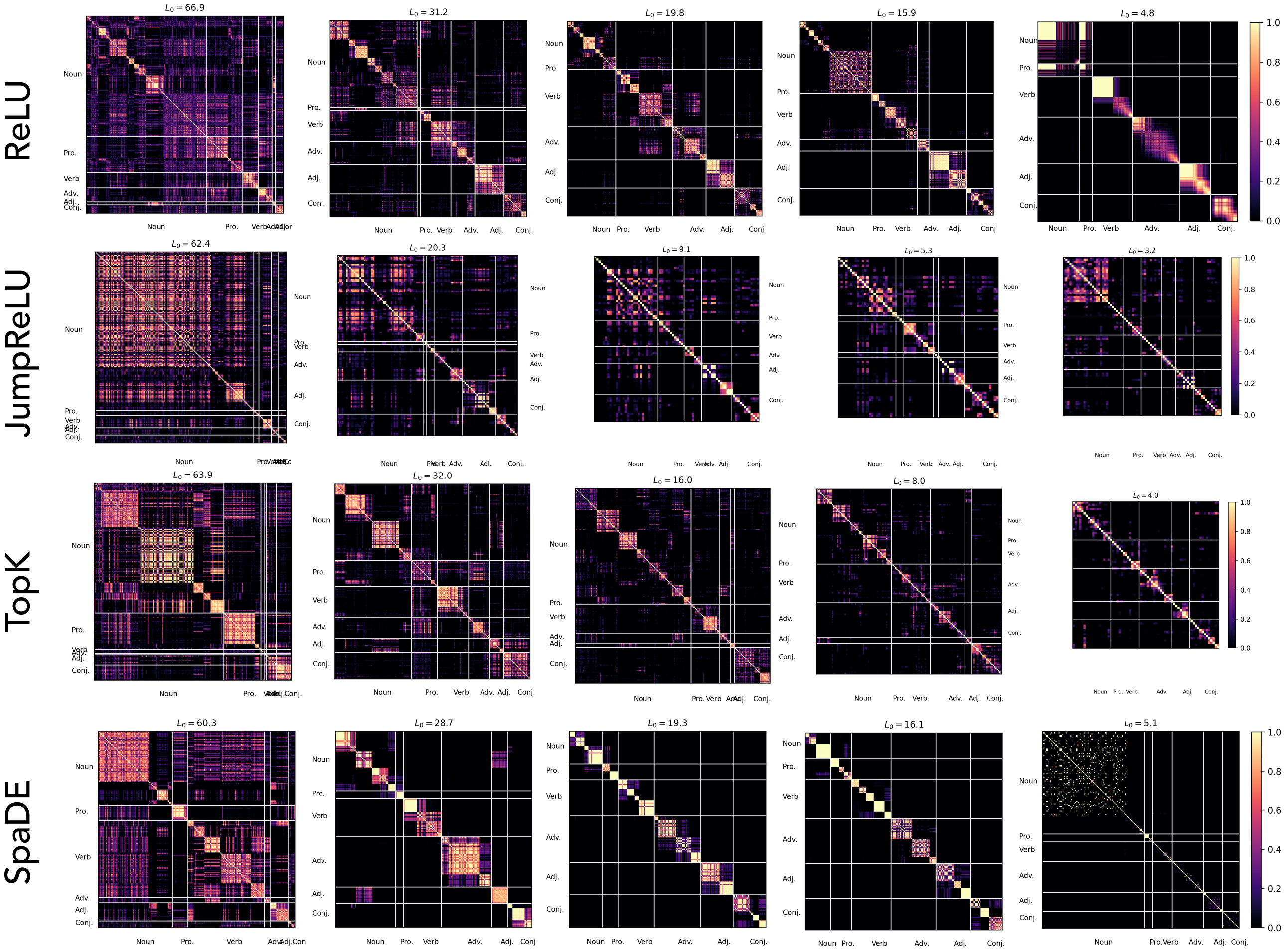

Figure F.21: Correlation between which datapoints a latent activates for in the Formal Language setup. Latents are sorted according to which concept (part-of-speech) they most strongly activated for (as measured using F1-score). White lines demarcate boundaries at which latents of different concepts start / end.

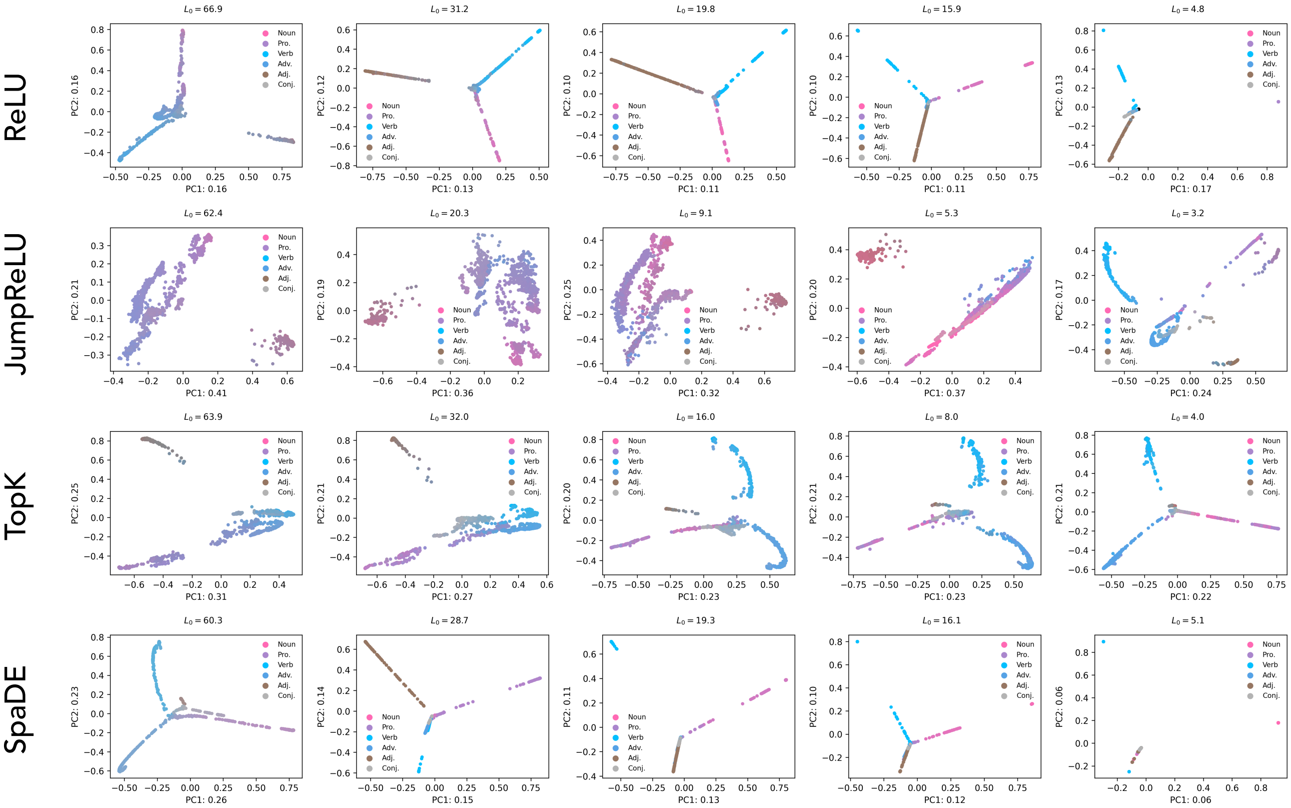

Figure F.22: 2D PCA visualization of sparse codes corresponding to different concepts (parts-of-speech).

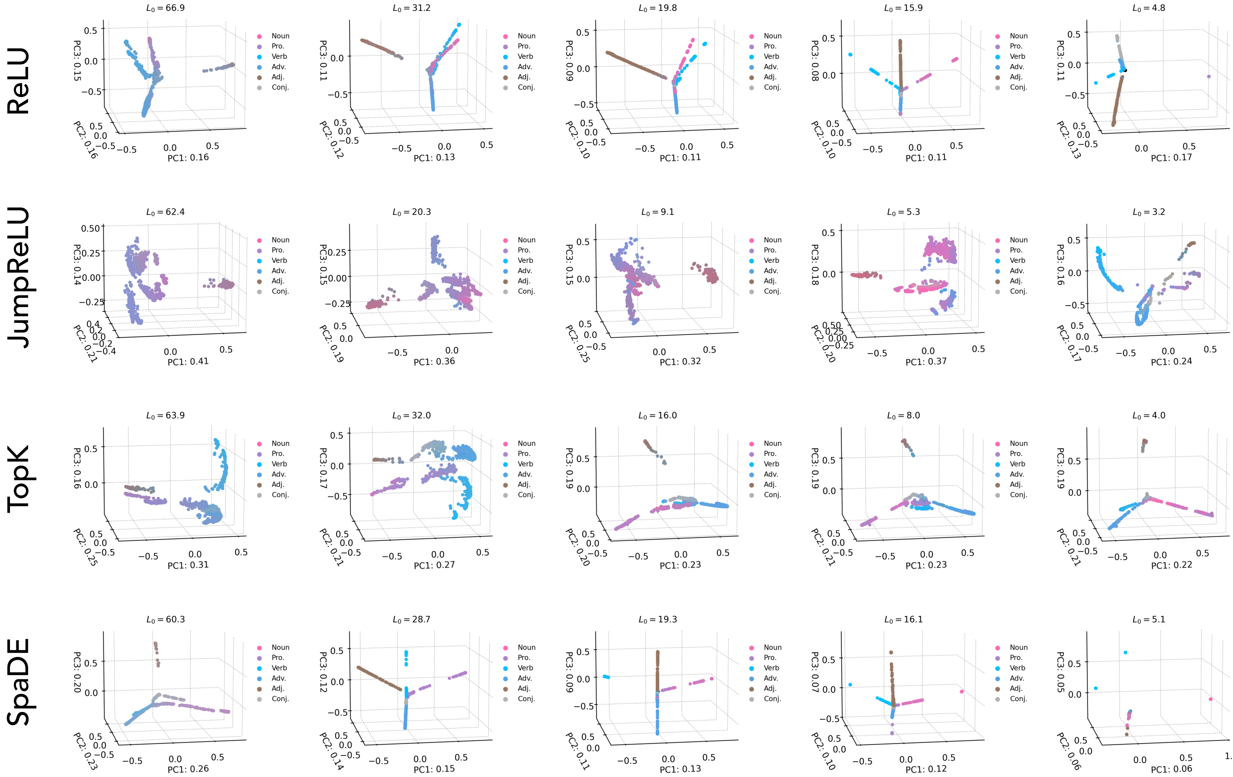

Figure F.23: 3D PCA visualization of sparse codes corresponding to different concepts (parts-of-speech).

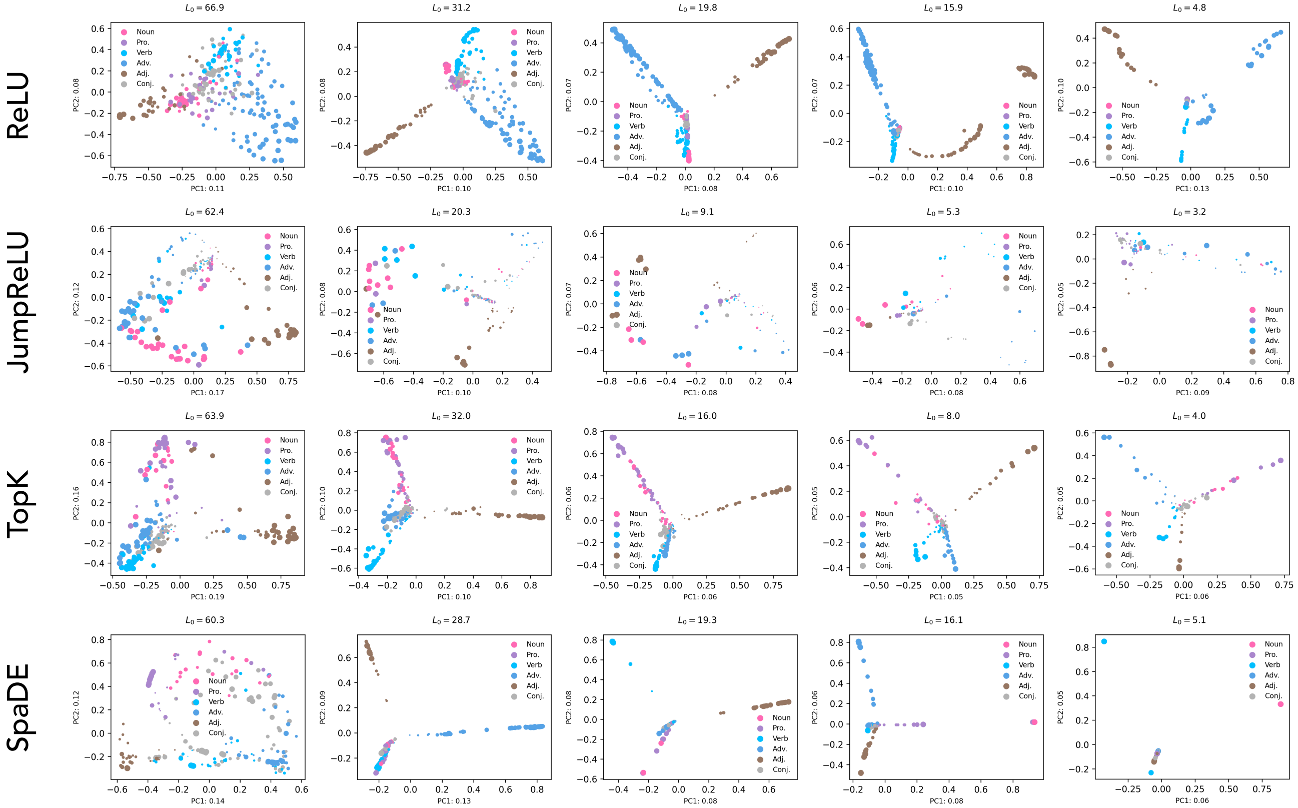

Figure F.24: 2D PCA visualization of a matrix whose elements capture which tokens a latent activates for. That is, which concepts (parts-of-speech) the latent is specialized towards, if any.

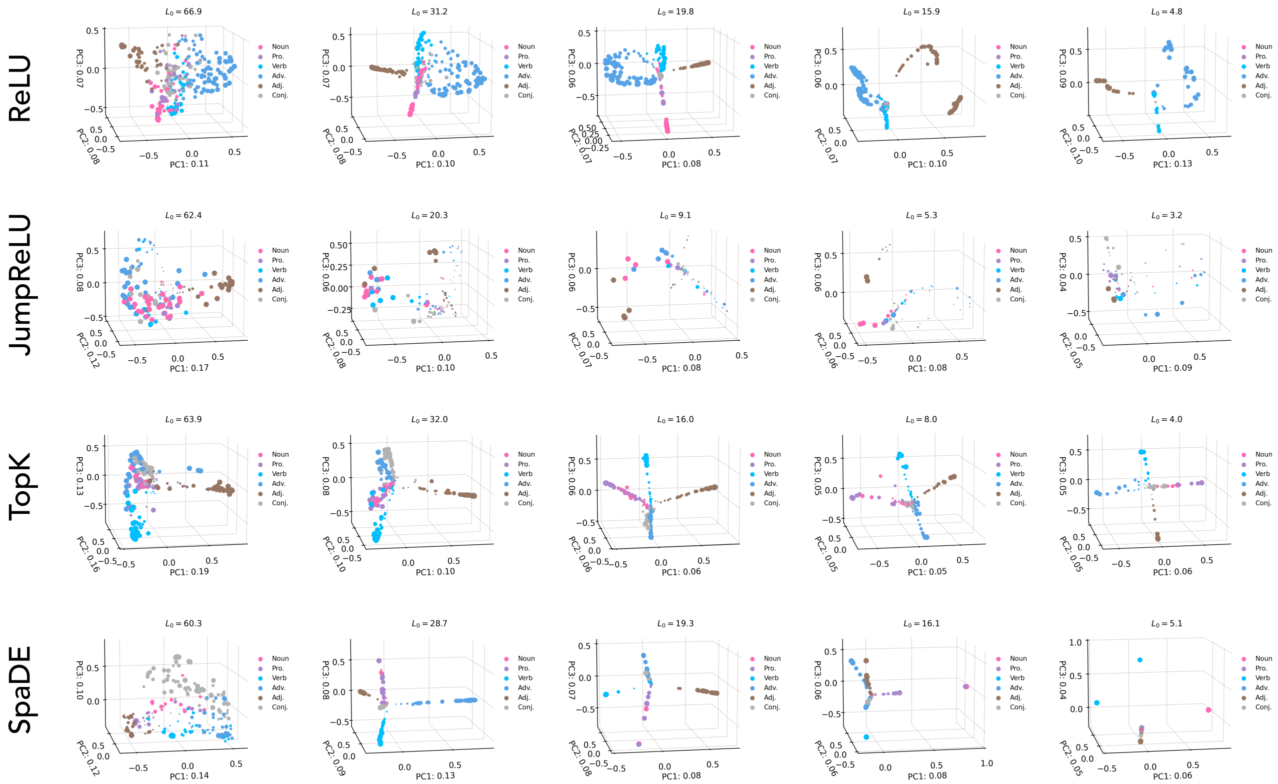

Figure F.25: 3D PCA visualization of a matrix whose elements capture which tokens a latent activates for. That is, which concepts (parts-of-speech) the latent is specialized towards, if any.

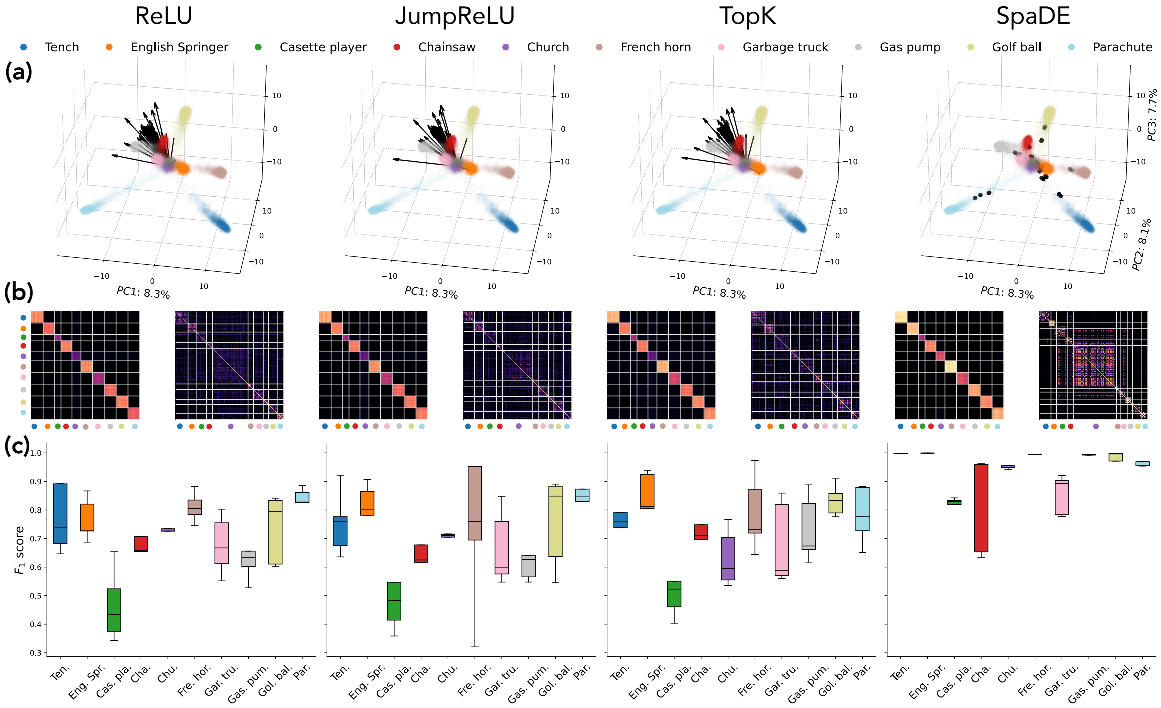

Figure F.26: **SAE properties on DINOv2 activations**. **(a)** 3-D PCA of model activations colored by class, and SAE encoder weights (points for SpaDE, arrows for other SAEs). **(b)** Cosine similarities of sparse codes of pairs of data and pairs of SAE latents (in order) for each SAE. White lines separate classes. **(c)** F_1 scores of top-5 most monosemantic latents for each SAE across classes (color-coded)

F.4. Vision Experiment

Dataset and Experiment: We use *Imagenette*, a 10-class subset of ImageNet (Deng et al., 2009), containing 1.5k images per class. Representations are extracted from the *DINOv2-base* (with registers), yielding 261 tokens per image. Over the course of 50 training epochs, this yields approximately 200 million tokens. SAEs are trained on all available tokens, including spatial, CLS, and registers tokens, for 50 epochs with 200 latent dimensions.

Observations: Results are shown in Fig. F.26. SpaDE again tiles the class structure well in the 3-D PCA (Row (a)). Similarities between sparse codes of data (first column of each SAE in Row (b)) show that all SAEs are able to decorrelate different classes in their latent representations. Latent co-occurrence (second column of each SAE in row (b)) is widespread in ReLU, JumpReLU and TopK SAEs, but it seems to be specific to certain pairs of latents in SpaDE. F_1 scores (row (c)) show that SpaDE has the most monosemantic latents across all classes. The varying F_1 scores for ReLU and JumpReLU across classes indicate different levels of linear separability across classes. Importantly, we find SpaDE identifies interpretable concepts such as foreground/background, different parts of objects in an image (hands, face, fins of fish, windows/ stairs in church images, eyes, ears, snout of dogs, etc), which are visualized using feature attribution maps in App. F.4.

We also show, visually, the concepts SpaDE has learnt in the vision experiment, by visualizing feature attribution maps for inputs from each class from Imagenette. We perform this visualization for the top concepts for each class for five classes- Tench (Fig. F.27), Chainsaw (Fig. F.28), Church (Fig. F.29), Golf (Fig. F.30) and Springer (Fig. F.31)).

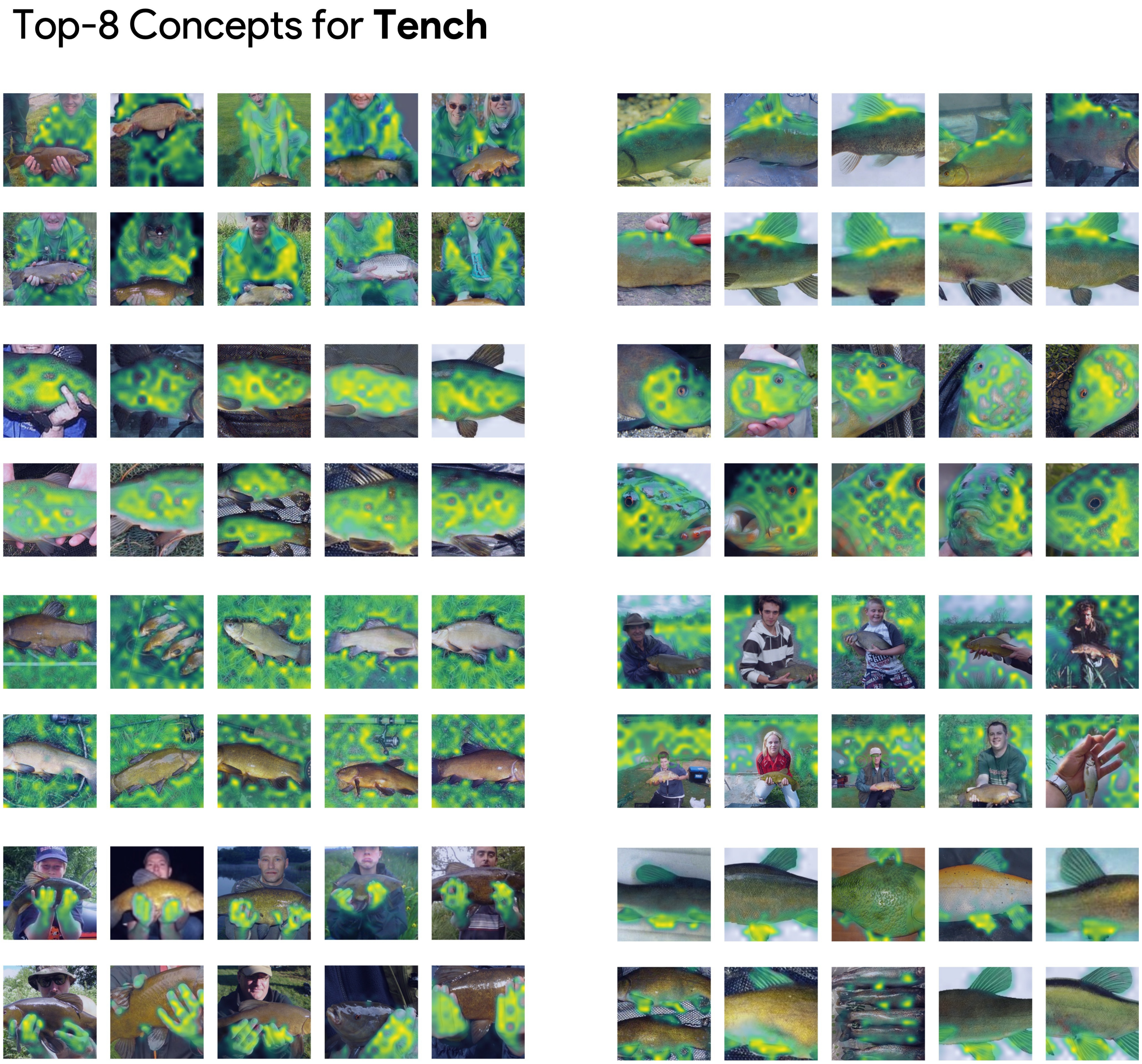

Figure F.27: Feature Attribution maps for monosemantic latents from SpaDE on the Tench class

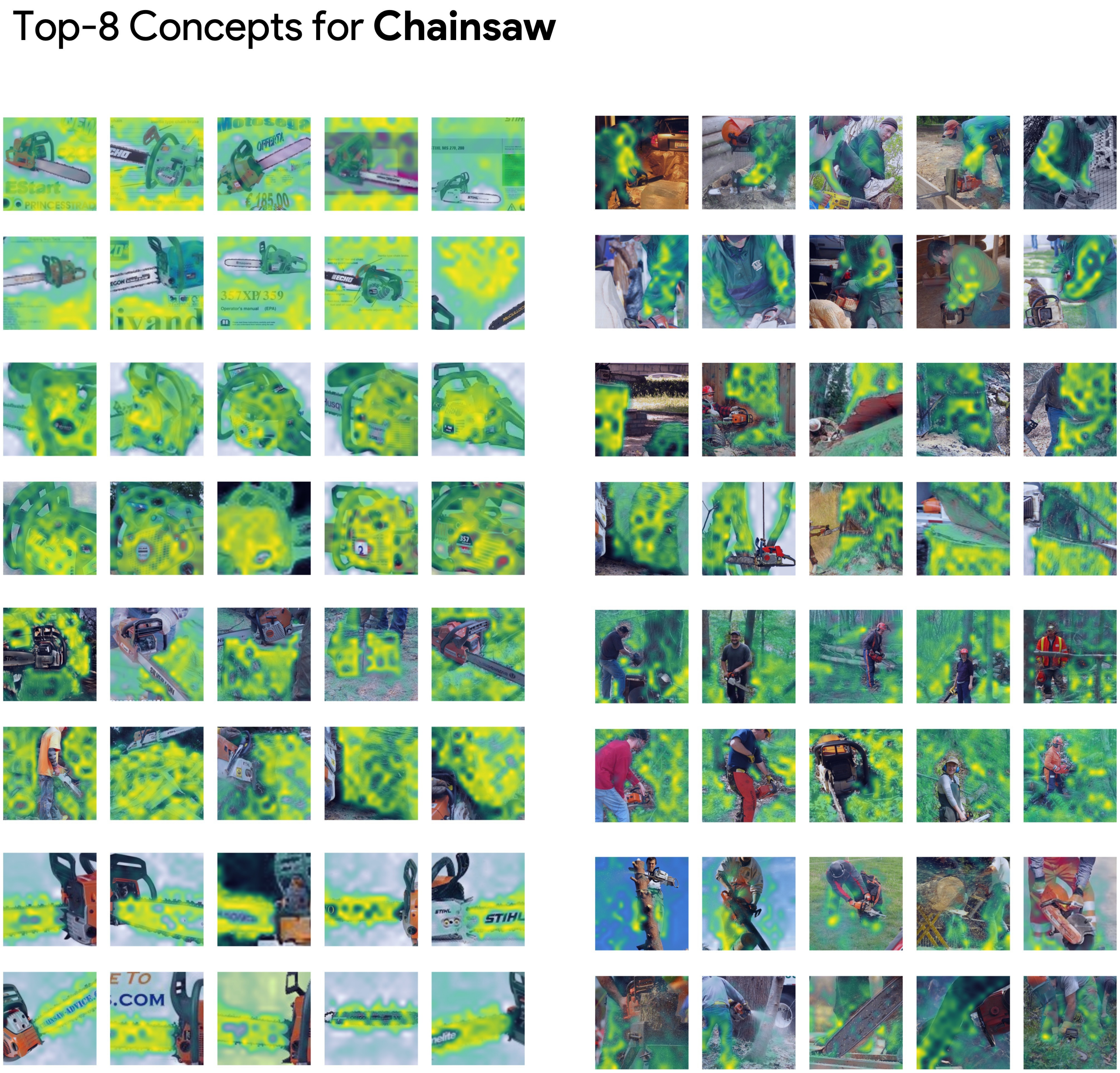

Figure F.28: Feature Attribution maps for monosemantic latents from SpaDE on the Chainsaw class

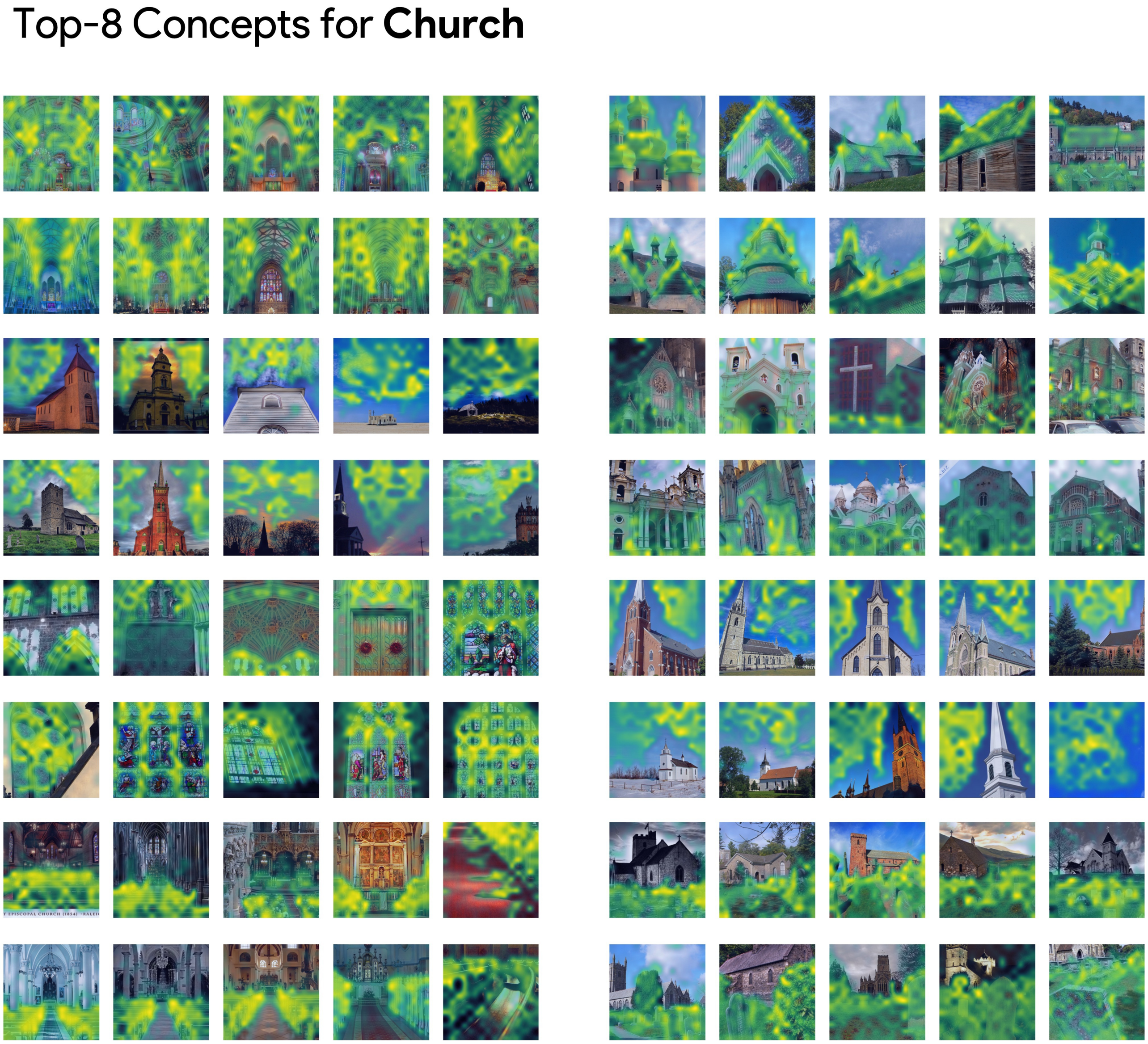

Figure F.29: Feature Attribution maps for monosemantic latents from SpaDE on the Church class

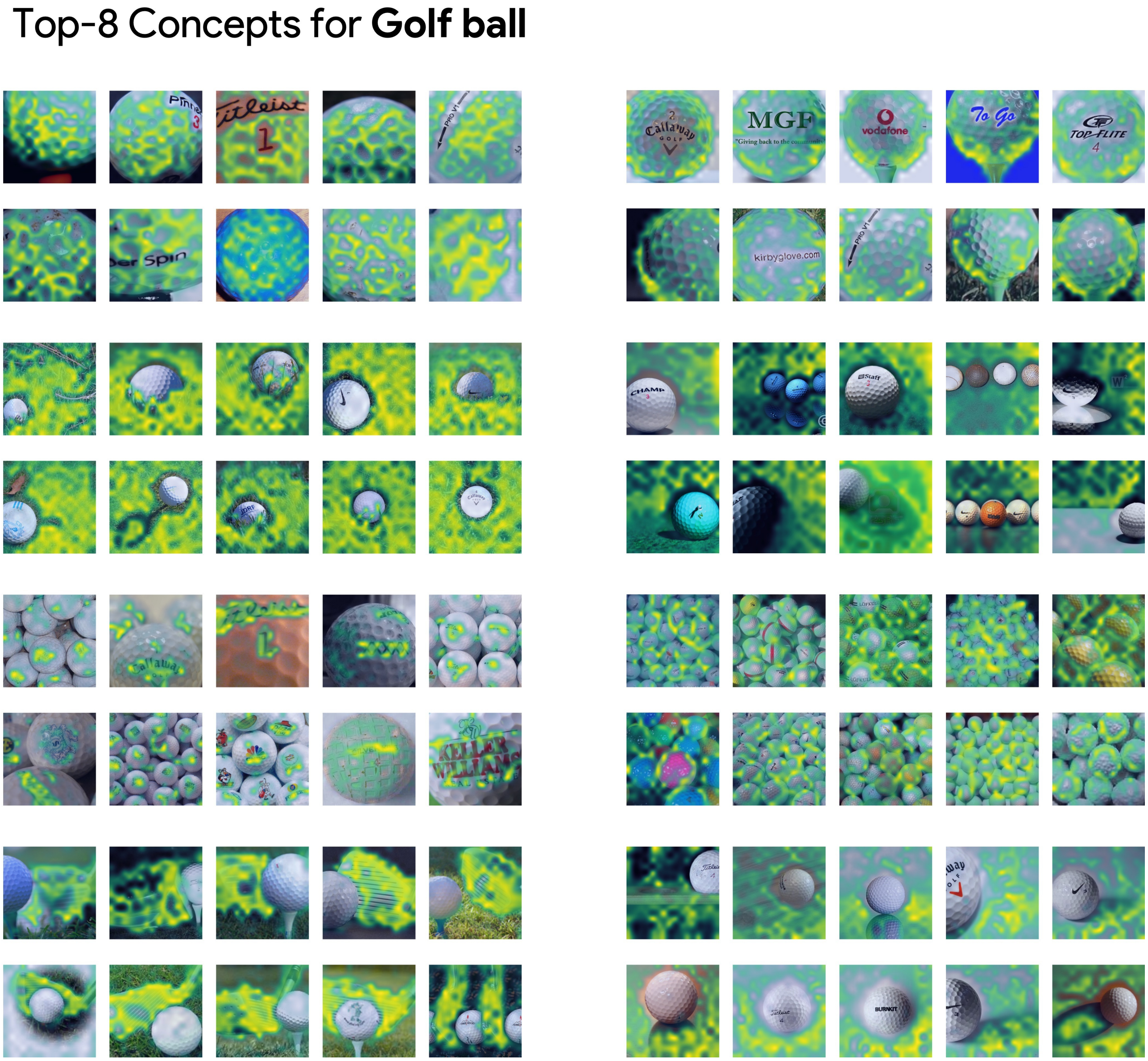

Figure F.30: Feature Attribution maps for monosemantic latents from SpaDE on the Golf class

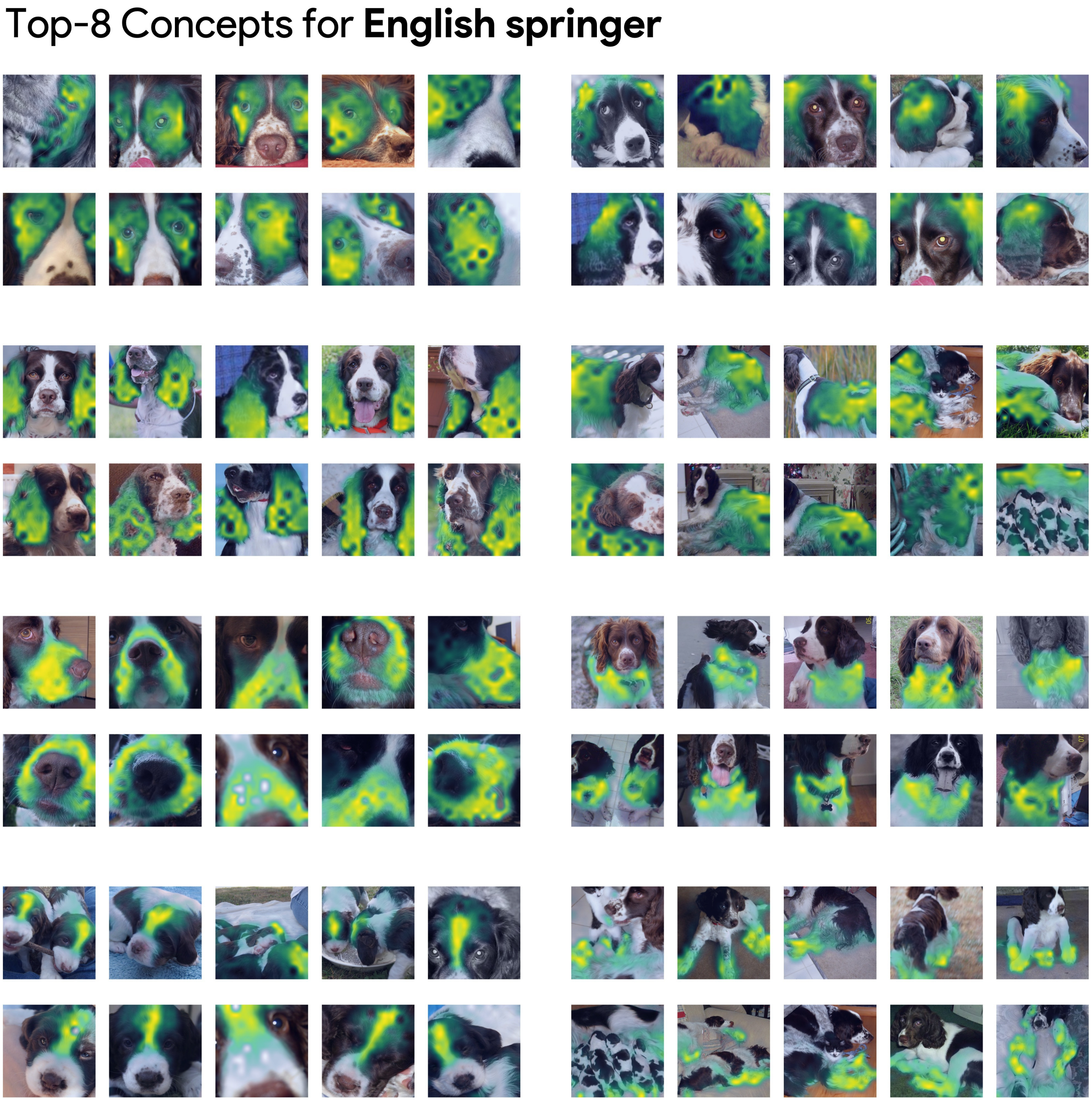

Figure F.31: Feature Attribution maps for monosemantic latents from SpaDE on the Springer class